K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: A2 **Roll No.: 1911027**

Experiment No. 08

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE: Disk Scheduling Algorithms

AIM: Implementation of Disk Scheduling Algorithm like FCFS, SSTF, SCAN, CSCAN, LOOK, CLOOK.

Expected Outcome of Experiment:

CO 4. To understand various Memory, I/O and File management techniques.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate "Operating Systems" McGraw Hill Third Edition.**
3. **William Stallings, "Operating System Internal & Design Principles", Pearson.**
4. **Andrew S. Tanenbaum, "Modern Operating System", Prentice Hall.**

Pre Lab/ Prior Concepts:

- Knowledge of disk scheduling algorithm.
- Calculation of seek time and transfer time etc.

Description of the application to be implemented:

First Come-First Serve (FCFS): FCFS is the simplest disk scheduling algorithm. As the name suggests, this algorithm entertains requests 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.

Algorithm:

- Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
- Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.
- Increment the total seek count with this distance.
- Currently serviced track position now becomes the new head position.
- Go to step 2 until all tracks in request array have not been serviced.

Advantages:

- In FCFS disk scheduling, there is no indefinite delay.
- There is no starvation in FCFS disk scheduling because each request gets a fair chance.

Disadvantages:

- FCFS scheduling is not offered as the best service.
- In FCFS, scheduling disk time is not optimized.

Implementation details: (printout of code)

Code:

```
print("--------------------------------------------------------------------
-----------------------------------------------")
print("-----------------------------------------------------------FCFS-------
-----------------------------------------------")
print("--------------------------------------------------------------------
-----------------------------------------------")
queries = input("Enter request sequence : ")
print("--------------------------------------------------------------------
-----------------------------------------------")
queries_list = list(queries.split())
queries_list = [int(i) for i in queries_list]
head = int(input("Enter initial head position : "))
print("--------------------------------------------------------------------
-----------------------------------------------")
totalHeadMovement = 0
print("Iniital Head Position : " + str(head))
seek_sequence=[]
print("--------------------------------------------------------------------
-----------------------------------------------")
for i in queries_list:
    print("Head position : ",head)
    print("Request : " + str(i))
    print("Head Movement : " + str(abs(i - head)))
    totalHeadMovement += abs(i - head)
    head = i
    print("New Head Position : " + str(head))
    seek_sequence.append(head)
    print("Seek Sequence : ",seek_sequence)
    print("--------------------------------------------------------------------
-------------------------------------------------")
print("Total Head Movement : " + str(totalHeadMovement))
print("--------------------------------------------------------------------
-----------------------------------------------")
```

Output:

```
----------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------FCFS----------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------
Enter request sequence : 176 79 34 60 92 11 41 114
----------------------------------------------------------------------------------------------------------------------------
Enter initial head position : 50
----------------------------------------------------------------------------------------------------------------------------
Iniital Head Position : 50
----------------------------------------------------------------------------------------------------------------------------
Head position :  50
Request : 176
Head Movement : 126
New Head Position : 176
Seek Sequence :  [176]
----------------------------------------------------------------------------------------------------------------------------
Head position :  176
Request : 79
Head Movement : 97
New Head Position : 79
Seek Sequence :  [176, 79]
----------------------------------------------------------------------------------------------------------------------------
Head position :  79
Request : 34
Head Movement : 45
New Head Position : 34
Seek Sequence :  [176, 79, 34]
----------------------------------------------------------------------------------------------------------------------------
Head position :  34
Request : 60
Head Movement : 26
New Head Position : 60
Seek Sequence :  [176, 79, 34, 60]
----------------------------------------------------------------------------------------------------------------------------
Head position :  60
Request : 92
Head Movement : 32
New Head Position : 92
Seek Sequence :  [176, 79, 34, 60, 92]
----------------------------------------------------------------------------------------------------------------------------
Head position :  92
Request : 11
Head Movement : 81
New Head Position : 11
Seek Sequence :  [176, 79, 34, 60, 92, 11]
----------------------------------------------------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------------------------------------------------------
Head position :  11
Request : 41
Head Movement : 30
New Head Position : 41
Seek Sequence :  [176, 79, 34, 60, 92, 11, 41]
----------------------------------------------------------------------------------------------------------------------------
Head position :  41
Request : 114
Head Movement : 73
New Head Position : 114
Seek Sequence :  [176, 79, 34, 60, 92, 11, 41, 114]
----------------------------------------------------------------------------------------------------------------------------
Total Head Movement : 510
----------------------------------------------------------------------------------------------------------------------------
```

Shortest Seek Time First (SSTF): 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. In SSTF 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.

Algorithm:

- Let Request array represents an array storing indexes of tracks that have been requested. 'head' is the position of disk head.
- Find the positive distance of all tracks in the request array from head.
- Find a track from requested array which has not been accessed/serviced yet and has minimum distance from head.
- Increment the total seek count with this distance.
- Currently serviced track position now becomes the new head position.
- Go to step 2 until all tracks in request array have not been serviced.

Advantages:

- Better performance than FCFS scheduling algorithm.
- It provides better throughput.
- This algorithm is used in Batch Processing system where throughput is more important.
- It has less average response and waiting time.

Disadvantages:

- Starvation is possible for some requests as it favours easy to reach request and ignores the far away processes.
- Their is lack of predictability because of high variance of response time.
- Switching direction slows things down.

Implementation details: (printout of code)

Code:

```
def smaller(initi):
    keys=[]
    values=[]
    for i in dic.keys():
        keys.append(i)
    for i in dic.values():
        values.append(i)
    small=values[0]
    val=-1
    for i,v in dic.items():
        if(v<=small and small!=-1):
            small=v
            val=i
    init=val
    dic.pop(val)
    return small,val,init
print("------------------------------------------------------------------------
-------------------------------------------------")
print("-----------------------------------------------------------SSTF-------
-------------------------------------------------")
print("------------------------------------------------------------------------
-------------------------------------------------")
inp=input("Enter request sequence : ").split()
print("------------------------------------------------------------------------
-------------------------------------------------")
initial_head=int(input("Enter initial head position : "))
print("------------------------------------------------------------------------
-------------------------------------------------")
req_seq=[]
dic={}
seek_sequence=[]
for i in inp:
    req_seq.append(int(i))
    if(initial_head>=int(i)):
        dic[int(i)]=initial_head-int(i)
    else:
        dic[int(i)]=int(i)-initial_head
print("Initial Head Position : ",initial_head)
```

```
print("-------------------------------------------------------------------
-------------------------------------------------")
print("Head position : ",initial_head)
temp,temp1,initial_head=smaller(initial_head)
seek_sequence.append(initial_head)
print("Request : ",temp1)
print("Head Movement : ",temp)
print("New Head Position : ",initial_head)
print("Seek Sequence : ",seek_sequence)
print("-------------------------------------------------------------------
-------------------------------------------------")
total_seek=0
total_seek=total_seek+temp
while(len(dic)!=0):
    print("Head position : ",initial_head)
    for i in dic.keys():
        if(initial_head>=int(i)):
            dic[i]=initial_head-int(i)
        else:
            dic[i]=int(i)-initial_head
    temp,temp1,initial_head=smaller(initial_head)
    total_seek=total_seek+temp
    seek_sequence.append(initial_head)
    print("Request : ",temp1)
    print("Head Movement : ",temp)
    print("New Head Position : ",initial_head)
    print("Seek Sequence : ",seek_sequence)
    print("-------------------------------------------------------------------
-------------------------------------------------")
print("Total head movement : ",total_seek)
print("-------------------------------------------------------------------
-------------------------------------------------")
```

Output:

```
--------------------------------------------------------------------------------------------------------------------
----------------------------------------------------SSTF----------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
Enter request sequence : 98 183 37 122 14 124 65 67
--------------------------------------------------------------------------------------------------------------------
Enter initial head position : 53
--------------------------------------------------------------------------------------------------------------------
Initial Head Position :  53
--------------------------------------------------------------------------------------------------------------------
Head position :  53
Request :  65
Head Movement :  12
New Head Position :  65
Seek Sequence :  [65]
--------------------------------------------------------------------------------------------------------------------
Head position :  65
Request :  67
Head Movement :  2
New Head Position :  67
Seek Sequence :  [65, 67]
--------------------------------------------------------------------------------------------------------------------
Head position :  67
Request :  37
Head Movement :  30
New Head Position :  37
Seek Sequence :  [65, 67, 37]
--------------------------------------------------------------------------------------------------------------------
Head position :  37
Request :  14
Head Movement :  23
New Head Position :  14
Seek Sequence :  [65, 67, 37, 14]
--------------------------------------------------------------------------------------------------------------------
Head position :  14
Request :  98
Head Movement :  84
New Head Position :  98
Seek Sequence :  [65, 67, 37, 14, 98]
--------------------------------------------------------------------------------------------------------------------
Head position :  98
Request :  122
Head Movement :  24
New Head Position :  122
Seek Sequence :  [65, 67, 37, 14, 98, 122]
--------------------------------------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------------------------------------------
Head position :  122
Request :  124
Head Movement :  2
New Head Position :  124
Seek Sequence :  [65, 67, 37, 14, 98, 122, 124]
--------------------------------------------------------------------------------------------------------------------
Head position :  124
Request :  183
Head Movement :  59
New Head Position :  183
Seek Sequence :  [65, 67, 37, 14, 98, 122, 124, 183]
--------------------------------------------------------------------------------------------------------------------
Total head movement :  236
--------------------------------------------------------------------------------------------------------------------
```

Elevator (SCAN): In Elevator disk scheduling algorithm, head starts from one end of the disk and moves towards the other end, servicing requests in between one by one and reach the other end. Then the direction of the head is reversed and the process continues as head continuously scan back and forth to access the disk. So, this algorithm works as an elevator and hence also known as the elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Algorithm:

- Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
- Let direction represents whether the head is moving towards left or right.
- In the direction in which head is moving service all tracks one by one.
- Calculate the absolute distance of the track from the head.
- Increment the total seek count with this distance.
- Currently serviced track position now becomes the new head position.
- Go to step 3 until we reach at one of the ends of the disk.
- If we reach at the end of the disk reverse the direction and go to step 2 until all tracks in request array have not been serviced.

Advantages:

- This algorithm is simple and easy to understand.
- SCAN algorithm have no starvation.
- This algorithm is better than FCFS Scheduling algorithm.

Disadvantages:

- More complex algorithm to implement.
- This algorithm is not fair because it cause long waiting time for the cylinders just visited by the head.
- It causes the head to move till the end of the disk in this way the requests arriving ahead of the arm position would get immediate service

but some other requests that arrive behind the arm position will have to wait for the request to complete.

Implementation details: (printout of code)

Code:

```
import random
def smaller(initi):
    for i in req_seq:
        if(i>=initi):
            req_seq.remove(i)
            return i-initi,i,i
    return range_var[1]-initi,range_var[1],range_var[1]
def greater(initi):
    for i in range(len(req_seq)-1,-1,-1):
        if(req_seq[i]<=initi):
            act=req_seq[i]
            req_seq.remove(req_seq[i])
            return initi-act,act,act
    return initi-range_var[0],range_var[0],range_var[0]
print("------------------------------------------------------------------
-----------------------------------------------")
print("-----------------------------------------------------------SCAN-------
-----------------------------------------------")
print("------------------------------------------------------------------
-----------------------------------------------")
inp=input("Enter range : ").split()
print("------------------------------------------------------------------
-----------------------------------------------")
range_var=[]
for i in inp:
    range_var.append(int(i))
inp=input("Enter request sequence : ").split()
print("------------------------------------------------------------------
-----------------------------------------------")
initial_head=int(input("Enter initial head position : "))
print("------------------------------------------------------------------
-----------------------------------------------")
req_seq=[]
for i in inp:
```

```
    req_seq.append(int(i))
req_seq.sort()
total_seek=0
n=int(input("Start from left(0) or right(1) : "))
print("--------------------------------------------------------------------
-----------------------------------------------")
seek_sequence=[]
if(n==1):
    while(initial_head!=range_var[1]):
        print("Head position : ",initial_head)
        temp,temp1,initial_head=smaller(initial_head)
        total_seek=total_seek+temp
        seek_sequence.append(initial_head)
        print("Request : ",temp1)
        print("Head Movement : ",temp)
        print("New Head Position : ",initial_head)
        print("Seek Sequence : ",seek_sequence)
        print("--------------------------------------------------------------
-------------------------------------------------------")
    while(initial_head!=range_var[0]):
        if(len(req_seq)==0):
            initial_head=range_var[0]
            break
        print("Head position : ",initial_head)
        temp,temp1,initial_head=initial_head-req_seq[-1],req_seq[-
1],req_seq[-1]
        req_seq.remove(initial_head)
        total_seek=total_seek+temp
        seek_sequence.append(initial_head)
        print("Request : ",temp1)
        print("Head Movement : ",temp)
        print("New Head Position : ",initial_head)
        print("Seek Sequence : ",seek_sequence)
        print("--------------------------------------------------------------
-------------------------------------------------------")
else:
    while(initial_head!=range_var[0]):
        print("Head position : ",initial_head)
        temp,temp1,initial_head=greater(initial_head)
        total_seek=total_seek+temp
        seek_sequence.append(initial_head)
```

```
        print("Request : ",temp1)
        print("Head Movement : ",temp)
        print("New Head Position : ",initial_head)
        print("Seek Sequence : ",seek_sequence)
        print("-------------------------------------------------------------
-----------------------------------------------------")
    while(initial_head!=range_var[1]):
        if(len(req_seq)==0):
            initial_head=range_var[1]
            break
        print("Head position : ",initial_head)
        temp,temp1,initial_head=req_seq[0]-
initial_head,req_seq[0],req_seq[0]
        req_seq.remove(initial_head)
        total_seek=total_seek+temp
        seek_sequence.append(initial_head)
        print("Request : ",temp1)
        print("Head Movement : ",temp)
        print("New Head Position : ",initial_head)
        print("Seek Sequence : ",seek_sequence)
        print("-------------------------------------------------------------
-----------------------------------------------------")
print("Total head movement : ",total_seek)
print("-------------------------------------------------------------------
-------------------------------------------------")
```

Output:

```
-------------------------------------------------------------------------------------------------------------------
-----------------------------------------------SCAN----------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
Enter range : 0 199
-------------------------------------------------------------------------------------------------------------------
Enter request sequence : 176 79 34 60 92 11 41 114
-------------------------------------------------------------------------------------------------------------------
Enter initial head position : 50
-------------------------------------------------------------------------------------------------------------------
Start from left(0) or right(1) : 0
-------------------------------------------------------------------------------------------------------------------
Head position :  50
Request :  41
Head Movement :  9
New Head Position :  41
Seek Sequence :  [41]
-------------------------------------------------------------------------------------------------------------------
Head position :  41
Request :  34
Head Movement :  7
New Head Position :  34
Seek Sequence :  [41, 34]
-------------------------------------------------------------------------------------------------------------------
Head position :  34
Request :  11
Head Movement :  23
New Head Position :  11
Seek Sequence :  [41, 34, 11]
-------------------------------------------------------------------------------------------------------------------
```

```
Head position :  11
Request :  0
Head Movement :  11
New Head Position :  0
Seek Sequence :  [41, 34, 11, 0]
---------------------------------------------------------------------------------------------------------------
Head position :  0
Request :  60
Head Movement :  60
New Head Position :  60
Seek Sequence :  [41, 34, 11, 0, 60]
---------------------------------------------------------------------------------------------------------------
Head position :  60
Request :  79
Head Movement :  19
New Head Position :  79
Seek Sequence :  [41, 34, 11, 0, 60, 79]
---------------------------------------------------------------------------------------------------------------
Head position :  79
Request :  92
Head Movement :  13
New Head Position :  92
Seek Sequence :  [41, 34, 11, 0, 60, 79, 92]
---------------------------------------------------------------------------------------------------------------
Head position :  92
Request :  114
Head Movement :  22
New Head Position :  114
Seek Sequence :  [41, 34, 11, 0, 60, 79, 92, 114]
---------------------------------------------------------------------------------------------------------------
Head position :  114
Request :  176
Head Movement :  62
New Head Position :  176
Seek Sequence :  [41, 34, 11, 0, 60, 79, 92, 114, 176]
---------------------------------------------------------------------------------------------------------------
Total head movement :  226
---------------------------------------------------------------------------------------------------------------
```

CSCAN: Circular SCAN (C-SCAN) scheduling algorithm is a modified version of SCAN disk scheduling algorithm that deals with the inefficiency of SCAN algorithm by servicing the requests more uniformly. Like SCAN (Elevator Algorithm) C-SCAN moves the head from one end servicing all the requests to the other end. However, as soon as the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip (see chart below) and starts servicing again once reaches the beginning. This is also known as the "Circular Elevator Algorithm" as it essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

Algorithm:

- Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.

- The head services only in the right direction from 0 to size of the disk.
- While moving in the left direction do not service any of the tracks.
- When we reach at the beginning(left end) reverse the direction.
- While moving in right direction it services all tracks one by one.
- While moving in right direction calculate the absolute distance of the track from the head.
- Increment the total seek count with this distance.
- Currently serviced track position now becomes the new head position.
- Go to step 6 until we reach at right end of the disk.
- If we reach at the right end of the disk reverse the direction and go to step 3 until all tracks in request array have not been serviced.

Advantages:

- The waiting time for the cylinders just visited by the head is reduced as compared to the SCAN Algorithm.
- It provides uniform waiting time.
- It provides better response time.

Disadvantages:

- It causes more seek movements as compared to SCAN Algorithm.
- It causes the head to move till the end of the disk even if there are no requests to be serviced.

Implementation details: (printout of code)

Code:

```
print("--------------------------------------------------------------------
-----------------------------------------------")
print("-----------------------------------------------------CSCAN-------
-----------------------------------------------")
print("--------------------------------------------------------------------
-----------------------------------------------")
ran = input("Enter range : ")
```

```
print("------------------------------------------------------------------------
-------------------------------------------------")
range_list = list(ran.split(" "))
range_list = [int(i) for i in range_list]
queries = input("Enter request sequence : ")
print("------------------------------------------------------------------------
-------------------------------------------------")
queries_list = list(queries.split(" "))
queries_list = [int(i) for i in queries_list]
head = int(input("Enter initital head position : "))
print("------------------------------------------------------------------------
-------------------------------------------------")
left_queries = list()
right_queries = list()
flag = input("Start from left(0) or right(1) : ")
print("------------------------------------------------------------------------
-------------------------------------------------")
left_queries.append(range_list[0])
right_queries.append(range_list[1])
for i in queries_list:
    if i < head:
        left_queries.append(i)
    else:
        right_queries.append(i)
left_queries.sort()
right_queries.sort()
if flag == "0":
    left_queries.reverse()
    right_queries.reverse()
totalHeadMovement = 0
seek_sequence = []
if flag == str(1):
    for i in right_queries:
        print("Head position : " + str(head))
        print("Request : " + str(i))
        print("Head Movement : " + str(abs(i - head)))
        totalHeadMovement += abs(i - head)
        head = i
        print("New Head Position : " + str(head))
        seek_sequence.append(head)
        print("Seek sequence : ", seek_sequence)
```

```
        print("------------------------------------------------------------
------------------------------------------------------")
    for i in left_queries:
        print("Head position : " + str(head))
        print("Request : " + str(i))
        print("Head Movement : " + str(abs(i - head)))
        totalHeadMovement += abs(i - head)
        head = i
        print("New Head Position : " + str(head))
        seek_sequence.append(head)
        print("Seek sequence : ", seek_sequence)
        print("------------------------------------------------------------
------------------------------------------------------")
elif flag == str(0):
    for i in left_queries:
        print("Head position : " + str(head))
        print("Request : " + str(i))
        print("Head Movement : " + str(abs(i - head)))
        totalHeadMovement += abs(i - head)
        head = i
        print("New Head Position : " + str(head))
        seek_sequence.append(head)
        print("Seek sequence : ", seek_sequence)
        print("------------------------------------------------------------
------------------------------------------------------")
    for i in right_queries:
        print("Head position : " + str(head))
        print("Request : " + str(i))
        print("Head Movement : " + str(abs(i - head)))
        totalHeadMovement += abs(i - head)
        head = i
        print("New Head Position : " + str(head))
        seek_sequence.append(head)
        print("Seek sequence : ", seek_sequence)
        print("------------------------------------------------------------
------------------------------------------------------")
print("Total Head Movement : " + str(totalHeadMovement))
print("------------------------------------------------------------------
-------------------------------------------------")
```

Output:

```
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------CSCAN-------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
Enter range : 0 199
-------------------------------------------------------------------------------------------------------------------------
Enter request sequence : 176 79 34 60 92 11 41 114
-------------------------------------------------------------------------------------------------------------------------
Enter initital head position : 50
-------------------------------------------------------------------------------------------------------------------------
Start from left(0) or right(1) : 1
-------------------------------------------------------------------------------------------------------------------------
Head position : 50
Request : 60
Head Movement : 10
New Head Position : 60
Seek sequence :  [60]
-------------------------------------------------------------------------------------------------------------------------
Head position : 60
Request : 79
Head Movement : 19
New Head Position : 79
Seek sequence :  [60, 79]
-------------------------------------------------------------------------------------------------------------------------
Head position : 79
Request : 92
Head Movement : 13
New Head Position : 92
Seek sequence :  [60, 79, 92]
-------------------------------------------------------------------------------------------------------------------------
Head position : 92
Request : 114
Head Movement : 22
New Head Position : 114
Seek sequence :  [60, 79, 92, 114]
-------------------------------------------------------------------------------------------------------------------------
Head position : 114
Request : 176
Head Movement : 62
New Head Position : 176
Seek sequence :  [60, 79, 92, 114, 176]
-------------------------------------------------------------------------------------------------------------------------
Head position : 176
Request : 199
Head Movement : 23
New Head Position : 199
Seek sequence :  [60, 79, 92, 114, 176, 199]
-------------------------------------------------------------------------------------------------------------------------
Head position : 199
Request : 0
Head Movement : 199
New Head Position : 0
Seek sequence :  [60, 79, 92, 114, 176, 199, 0]
-------------------------------------------------------------------------------------------------------------------------
Head position : 0
Request : 11
Head Movement : 11
New Head Position : 11
Seek sequence :  [60, 79, 92, 114, 176, 199, 0, 11]
-------------------------------------------------------------------------------------------------------------------------
Head position : 11
Request : 34
Head Movement : 23
New Head Position : 34
Seek sequence :  [60, 79, 92, 114, 176, 199, 0, 11, 34]
-------------------------------------------------------------------------------------------------------------------------
Head position : 34
Request : 41
Head Movement : 7
New Head Position : 41
Seek sequence :  [60, 79, 92, 114, 176, 199, 0, 11, 34, 41]
-------------------------------------------------------------------------------------------------------------------------
Total Head Movement : 389
-------------------------------------------------------------------------------------------------------------------------
```

LOOK: LOOK is the advanced version of SCAN (elevator) disk scheduling algorithm which gives slightly better seek time than any other algorithm in the hierarchy (FCFS->SRTF->SCAN->C-SCAN->LOOK). The LOOK algorithm services request similarly as SCAN algorithm meanwhile it also "looks" ahead as if there are more tracks that are needed to be serviced in the same direction. If there are no pending requests in the moving direction the head reverses the direction and start servicing requests in the opposite direction. The main reason behind the better performance of LOOK algorithm in comparison to SCAN is because in this algorithm the head is not allowed to move till the end of the disk.

Algorithm:

- Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
- The initial direction in which head is moving is given and it services in the same direction.
- The head services all the requests one by one in the direction head is moving.
- The head continues to move in the same direction until all the request in this direction are finished.
- While moving in this direction calculate the absolute distance of the track from the head.
- Increment the total seek count with this distance.
- Currently serviced track position now becomes the new head position.
- Go to step 5 until we reach at last request in this direction.
- If we reach where no requests are needed to be serviced in this direction reverse the direction and go to step 3 until all tracks in request array have not been serviced.

Advantages:

- It does not causes the head to move till the ends of the disk when there are no requests to be serviced.
- It provides better performance as compared to SCAN Algorithm.

- It does not lead to starvation.
- It provides low variance in response time and waiting time.

Disadvantages:

- There is an overhead of finding the end requests.
- It causes long waiting time for the cylinders just visited by the head.

Implementation details: (printout of code)

Code:

```
print("--------------------------------------------------------------------
-------------------------------------------------")
print("-------------------------------------------------------------LOOK-------
-------------------------------------------------")
print("--------------------------------------------------------------------
-------------------------------------------------")
ran = input("Enter range : ")
print("--------------------------------------------------------------------
-------------------------------------------------")
range_list = list(ran.split(" "))
range_list = [int(i) for i in range_list]
queries = input("Enter request sequence : ")
print("--------------------------------------------------------------------
-------------------------------------------------")
queries_list = list(queries.split(" "))
queries_list = [int(i) for i in queries_list]
head = int(input("Enter initial head position : "))
print("--------------------------------------------------------------------
-------------------------------------------------")
left_queries = list()
right_queries = list()
flag = input("Start from left(0) or right(1) : ")
print("--------------------------------------------------------------------
-------------------------------------------------")
for i in queries_list:
    if i < head:
        left_queries.append(i)
    else:
        right_queries.append(i)
```

```
left_queries.sort()
right_queries.sort()
left_queries.reverse()
totalHeadMovement = 0
seek_sequence = []
if flag == "1":
    for i in right_queries:
        print("Head position : " + str(head))
        print("Request : " + str(i))
        print("Head Movement : " + str(abs(i - head)))
        totalHeadMovement += abs(i - head)
        head = i
        print("New Head Position : " + str(head))
        seek_sequence.append(head)
        print("Seek sequence : ", seek_sequence)
        print("------------------------------------------------------------
------------------------------------------------------")
    for i in left_queries:
        print("Head position : " + str(head))
        print("Request : " + str(i))
        print("Head Movement : " + str(abs(i - head)))
        totalHeadMovement += abs(i - head)
        head = i
        print("New Head Position : " + str(head))
        seek_sequence.append(head)
        print("Seek sequence : ", seek_sequence)
        print("------------------------------------------------------------
------------------------------------------------------")
elif flag == "0":
    for i in left_queries:
        print("Head position : " + str(head))
        print("Request : " + str(i))
        print("Head Movement : " + str(abs(i - head)))
        totalHeadMovement += abs(i - head)
        head = i
        print("New Head Position : " + str(head))
        seek_sequence.append(head)
        print("Seek sequence : ", seek_sequence)
        print("------------------------------------------------------------
------------------------------------------------------")
    for i in right_queries:
```

```
        print("Head position : " + str(head))
        print("Request : " + str(i))
        print("Head Movement : " + str(abs(i - head)))
        totalHeadMovement += abs(i - head)
        head = i
        print("New Head Position : " + str(head))
        seek_sequence.append(head)
        print("Seek sequence : ", seek_sequence)
        print("------------------------------------------------------------
------------------------------------------------------------")
print("Total Head Movement : " + str(totalHeadMovement))
print("------------------------------------------------------------------
------------------------------------------------")
```

Output:

```
--------------------------------------------------------------------------------------------------------------------
----------------------------------------LOOK----------------------------------------
--------------------------------------------------------------------------------------------------------------------
Enter range : 0 199
--------------------------------------------------------------------------------------------------------------------
Enter request sequence : 176 79 34 60 92 11 41 114
--------------------------------------------------------------------------------------------------------------------
Enter initial head position : 50
--------------------------------------------------------------------------------------------------------------------
Start from left(0) or right(1) : 1
--------------------------------------------------------------------------------------------------------------------
Head position : 50
Request : 60
Head Movement : 10
New Head Position : 60
Seek sequence :  [60]
--------------------------------------------------------------------------------------------------------------------
Head position : 60
Request : 79
Head Movement : 19
New Head Position : 79
Seek sequence :  [60, 79]
--------------------------------------------------------------------------------------------------------------------
Head position : 79
Request : 92
Head Movement : 13
New Head Position : 92
Seek sequence :  [60, 79, 92]
--------------------------------------------------------------------------------------------------------------------
Head position : 92
Request : 114
Head Movement : 22
New Head Position : 114
Seek sequence :  [60, 79, 92, 114]
--------------------------------------------------------------------------------------------------------------------
Head position : 114
Request : 176
Head Movement : 62
New Head Position : 176
Seek sequence :  [60, 79, 92, 114, 176]
--------------------------------------------------------------------------------------------------------------------
```

```
Head position : 176
Request : 41
Head Movement : 135
New Head Position : 41
Seek sequence :  [60, 79, 92, 114, 176, 41]
Head position : 41
Request : 34
Head Movement : 7
New Head Position : 34
Seek sequence :  [60, 79, 92, 114, 176, 41, 34]
Head position : 34
Request : 11
Head Movement : 23
New Head Position : 11
Seek sequence :  [60, 79, 92, 114, 176, 41, 34, 11]
Total Head Movement : 291
```

CLOOK: C-LOOK is an enhanced version of both SCAN as well as LOOK disk scheduling algorithms. This algorithm also uses the idea of wrapping the tracks as a circular cylinder as C-SCAN algorithm but the seek time is better than C-SCAN algorithm. We know that C-SCAN is used to avoid starvation and services all the requests more uniformly, the same goes for C-LOOK. In this algorithm, the head services requests only in one direction(either left or right) until all the requests in this direction are not serviced and then jumps back to the farthest request on the other direction and service the remaining requests which gives a better uniform servicing as well as avoids wasting seek time for going till the end of the disk.

Algorithm:

- Let Request array represents an array storing indexes of the tracks that have been requested in ascending order of their time of arrival and head is the position of the disk head.
- The initial direction in which the head is moving is given and it services in the same direction.
- The head services all the requests one by one in the direction it is moving.
- The head continues to move in the same direction until all the requests in this direction have been serviced.
- While moving in this direction, calculate the absolute distance of the tracks from the head.

- Increment the total seek count with this distance.
- Currently serviced track position now becomes the new head position.
- Go to step 5 until we reach the last request in this direction.
- If we reach the last request in the current direction then reverse the direction and move the head in this direction until we reach the last request that is needed to be serviced in this direction without servicing the intermediate requests.
- Reverse the direction and go to step 3 until all the requests have not been serviced.

Advantages:

- In C-LOOK the head does not have to move till the end of the disk if there are no requests to be serviced.
- There is less waiting time for the cylinders which are just visited by the head in C-LOOK.
- C-LOOK provides better performance when compared to LOOK Algorithm.
- Starvation is avoided in C-LOOK.
- Low variance is provided in waiting time and response time.

Disadvantages:

- Extra code or effort to build this algorithm.
- It should not be used in case there is very high load.

Implementation details: (printout of code)

Code:

```
import random
def smaller(initi):
    for i in req_seq:
        if(i>=initi):
            req_seq.remove(i)
            return i-initi,i,i
    if(len(req_seq)!=0):
```

```
        va=req_seq[0]
        req_seq.remove(req_seq[0])
        return initi-va,va,va
    else:
        return "X","X","X"
def greater(initi):
    for i in range(len(req_seq)-1,-1,-1):
        if(req_seq[i]<=initi):
            act=req_seq[i]
            req_seq.remove(req_seq[i])
            return initi-act,act,act
    if(len(req_seq)!=0):
        va=req_seq[len(req_seq)-1]
        req_seq.remove(req_seq[len(req_seq)-1])
        return va-initi,va,va
    else:
        return "X","X","X"
print("------------------------------------------------------------------------
-----------------------------------------------")
print("-------------------------------------------------------------CLOOK-------
-----------------------------------------------")
print("------------------------------------------------------------------------
-----------------------------------------------")
inp=input("Input range : ").split()
print("------------------------------------------------------------------------
-----------------------------------------------")
range_var=[]
for i in inp:
    range_var.append(int(i))
inp=input("Enter request sequence : ").split()
print("------------------------------------------------------------------------
-----------------------------------------------")
initial_head=int(input("Enter initial head position : "))
print("------------------------------------------------------------------------
-----------------------------------------------")
req_seq=[]
for i in inp:
    req_seq.append(int(i))
req_seq.sort()
total_seek=0
n=int(input("Start from left(0) or right(1) : "))
```

```
print("------------------------------------------------------------------
-------------------------------------------------")
seek_sequence=[]
if(n==1):
    while(len(req_seq)!=0):
        print("Head position : ",initial_head)
        temp,temp1,initial_head=smaller(initial_head)
        if(temp=="X"):
            break
        seek_sequence.append(initial_head)
        print("Request : ",temp1)
        print("Head Movement : ",temp)
        print("New Head Position : ",initial_head)
        print("Seek Sequence : ",seek_sequence)
        print("------------------------------------------------------------
-----------------------------------------------------")
        total_seek=total_seek+temp
else:
    while(len(req_seq)!=0):
        print("Head position : ",initial_head)
        temp,temp1,initial_head=greater(initial_head)
        if(temp=="X"):
            break
        seek_sequence.append(initial_head)
        print("Request : ",temp1)
        print("Head Movement : ",temp)
        print("New Head Position : ",initial_head)
        print("Seek Sequence : ",seek_sequence)
        print("------------------------------------------------------------
-----------------------------------------------------")
        total_seek=total_seek+temp
print("Total head movement : ",total_seek)
print("------------------------------------------------------------------
-------------------------------------------------")
```

Output:

```
----------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------CLOOK---------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------
Input range : 0 199
----------------------------------------------------------------------------------------------------------------------------
Enter request sequence : 176 79 34 60 92 11 41 114
----------------------------------------------------------------------------------------------------------------------------
Enter initial head position : 50
----------------------------------------------------------------------------------------------------------------------------
Start from left(0) or right(1) : 1
----------------------------------------------------------------------------------------------------------------------------
Head position :  50
Request :  60
Head Movement :  10
New Head Position :  60
Seek Sequence :  [60]
----------------------------------------------------------------------------------------------------------------------------
Head position :  60
Request :  79
Head Movement :  19
New Head Position :  79
Seek Sequence :  [60, 79]
----------------------------------------------------------------------------------------------------------------------------
Head position :  79
Request :  92
Head Movement :  13
New Head Position :  92
Seek Sequence :  [60, 79, 92]
----------------------------------------------------------------------------------------------------------------------------
Head position :  92
Request :  114
Head Movement :  22
New Head Position :  114
Seek Sequence :  [60, 79, 92, 114]
----------------------------------------------------------------------------------------------------------------------------
Head position :  114
Request :  176
Head Movement :  62
New Head Position :  176
Seek Sequence :  [60, 79, 92, 114, 176]
----------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------
Head position :  176
Request :  11
Head Movement :  165
New Head Position :  11
Seek Sequence :  [60, 79, 92, 114, 176, 11]
----------------------------------------------------------------------------------------------------------------------------
Head position :  11
Request :  34
Head Movement :  23
New Head Position :  34
Seek Sequence :  [60, 79, 92, 114, 176, 11, 34]
----------------------------------------------------------------------------------------------------------------------------
Head position :  34
Request :  41
Head Movement :  7
New Head Position :  41
Seek Sequence :  [60, 79, 92, 114, 176, 11, 34, 41]
----------------------------------------------------------------------------------------------------------------------------
Total head movement :  321
----------------------------------------------------------------------------------------------------------------------------
```

Conclusion: **Understood the concept of disk scheduling in operating system also learned about various disk scheduling algorithm like FCFS, SSTF, SCAN, CSCAN, LOOK and CLOOK. Implemented all the mentioned algorithms using python.**

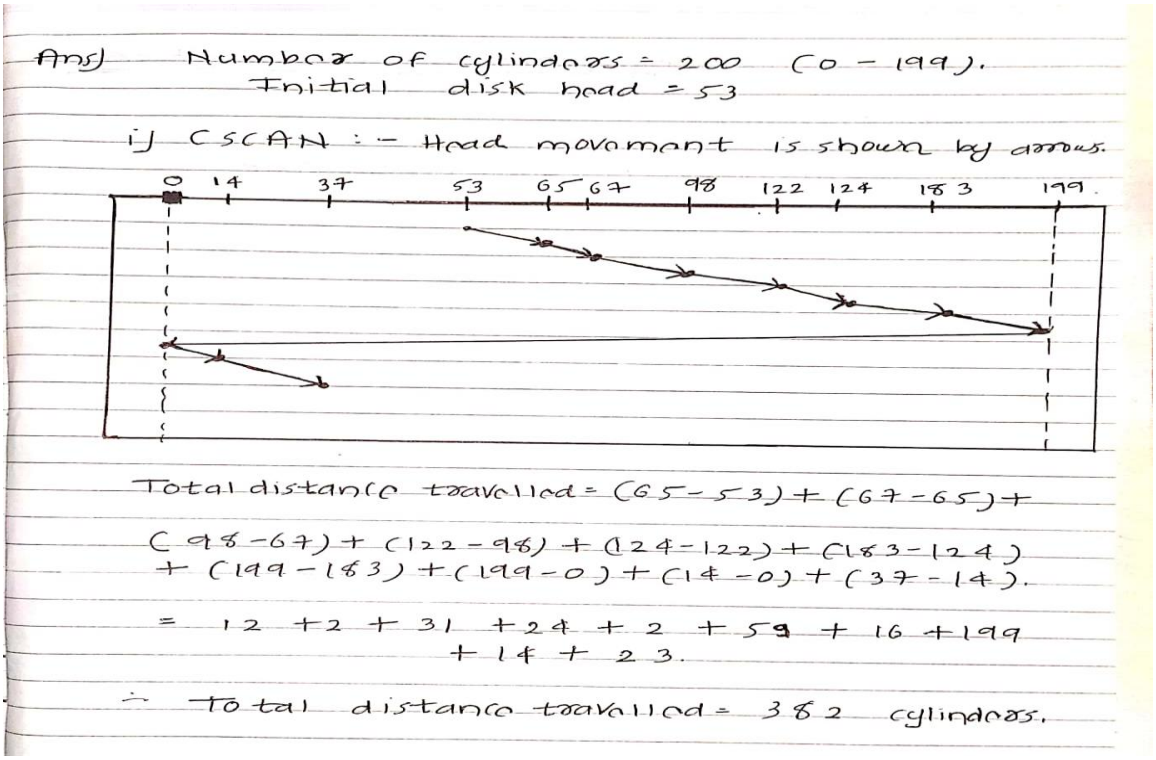
Post Lab Descriptive Questions

1. A disk drive has 200 cylinders numbered from 0 to 199. The disk head is initially at cylinder 53. The queue of pending requests in FIFO order is :

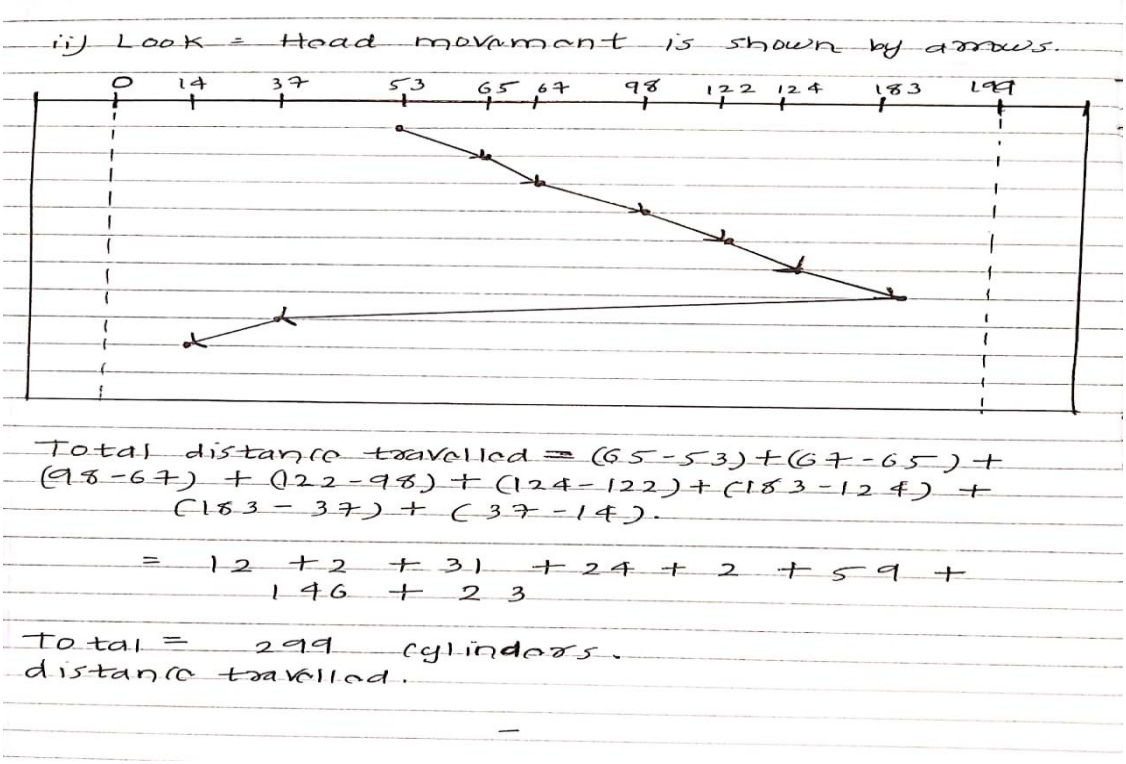
   98, 183, 37, 122, 14, 124, 65, 67.

   Starting from the current head position, what is the total distance travelled (in cylinders) by disk arm to satisfy the requests using CSCAN and Look. Illustrate with figures in each case.

ANS)



K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering



Post Lab Objective Questions

1. In a hard disk, what rotates about a central spindle _____
   a. Disk
   b. Platter
   c. Sector
   d. None of the above
   ANS: b. Platter

2. The time required to move the disk arm to the required track is known as _____
   a. Latency time
   b. Access time
   c. Seek time
   d. None of the above
   ANS: c. Seek time

Date: 24 / 11 / 2021 **Signature of faculty in-charge**