Project

Operating System and System Programming

15B11CI412

Submitted by:

Jayant Goel (18103255) B8

Under the supervision of:

Dr. Alka Singhal & Dr. Taj Alam

Implementation and Performance comparison of four disk scheduling algorithms

Hard disks are being employed to store vast info information in all modern computers. Disk drives should give quicker access time so as to optimize the speed of I/O operations. In a multitasking system with several processes, disk performance can be improved by incorporating a scheduling algorithm for maintaining several unfinished requests within the disk queue.

The Four disk scheduling algorithms (FCFS, SSTF, LOOK for each upward and downward direction, and C-LOOK) to measure their performance in terms of total head movement. The developed machine runs with success in a concurrent execution surroundings and therefore the results demonstrate that LOOK (downward direction) the algorithm provides the most effective results for given take a look at samples due to the reduction of an oversized range of unneeded head movements. As several wild swings are experienced by FCFS scheme so it gives the worst scheduling performance. SSTF is far higher compared to seem (upward direction) and C-LOOK. It's has been noticed that LOOK is a lot of economical than C-LOOK at all hundreds whereas C-LOOK is best at high hundreds solely because it reduces the starvation problem.

The performance of every algorithm, however, heavily depends on the amount and sort of requests.

DISK SCHEDULING

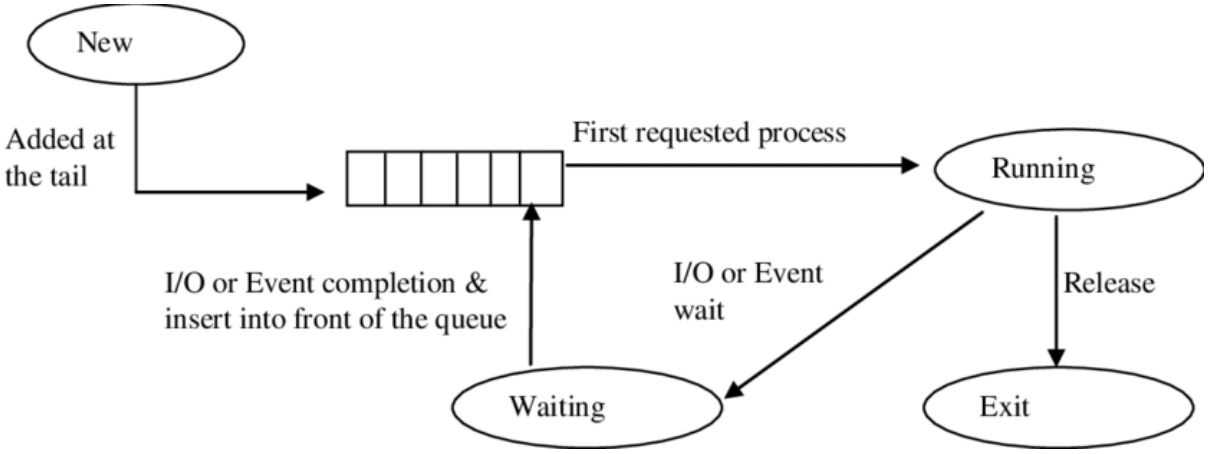
Since most jobs depend heavily on the disk for loading and I/O operations, it is important that disk service be as fast as possible. Disk speed is composed of three parts: **seek time, latency time, and transfer time**. Every I/O device, including each disk drive, has a queue of pending requests. Whenever a process needs I/O to or from the disk, it issues a system call to the operating system. This request specifies several pieces of information:
(1) Type of I/O operations,

(2) Address of disk (drive, cylinder, surface, and block),
(3) Address of memory, and
(4) Amount of information is to be transferred (a byte or word count).

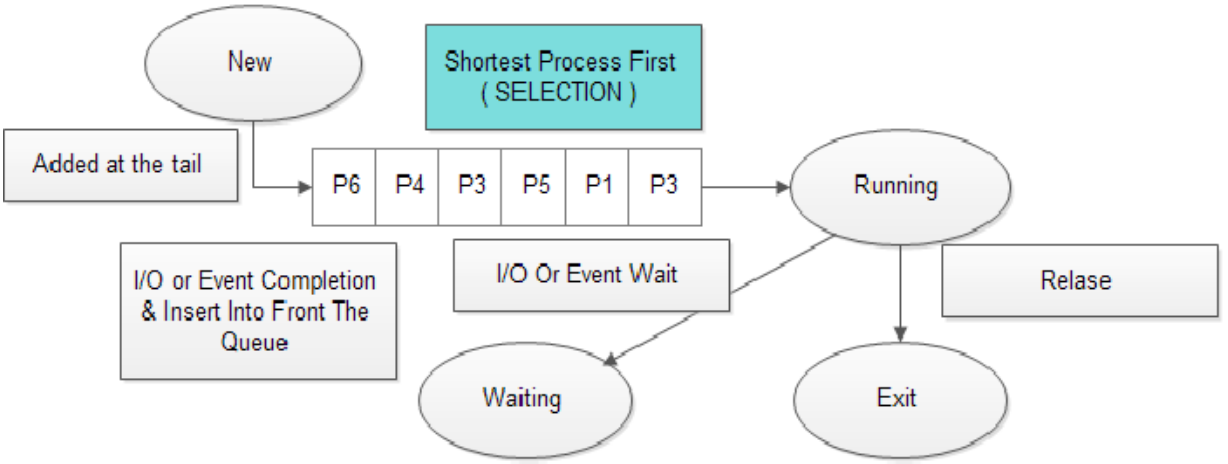
If the desired disk drive and controller are available, the request can be serviced immediately. However, while the drive or controller is serving one request, any additional requests will need to be queued. For a multiprogramming system with many processes, the disk queue may often be nonempty. Thus, when a request is complete, a new request is picked from the queue and it is serviced. A disk service requires that the head be moved to the desired track; it must wait for latency and seek time, and finally transfer the data to memory.

Different algorithms such as **FCFS, SSTF, LOOK and CLOOK** are used for selecting request for servicing from the queue of requests.

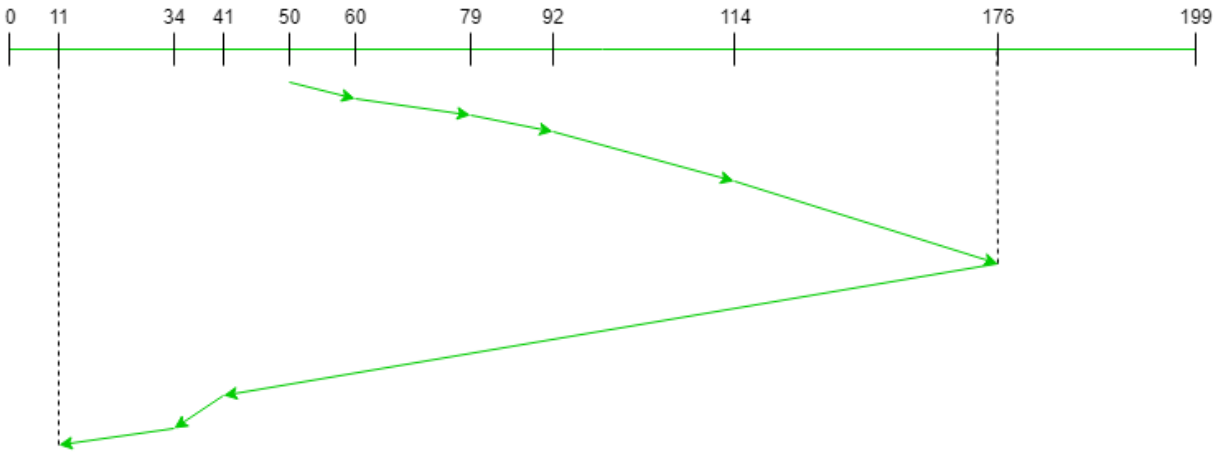
First-Come, First-Served (FCFS) is the simplest form of disk scheduling. This algorithm is easy to implement using FIFO queue. Wild swing of FCFS can increase total head movements.



Shortest-Seek-Time-First (SSTF) seems reasonable to service together all requests close to the current head position, before moving the head far away to service another request. It selects the request with the minimum seek time from the current head position. Since the seek time is generally proportional to the track difference between the requests, it is implemented by moving the head to the closest track in the request queue. Some of the requests may wait indefinitely in this approach.

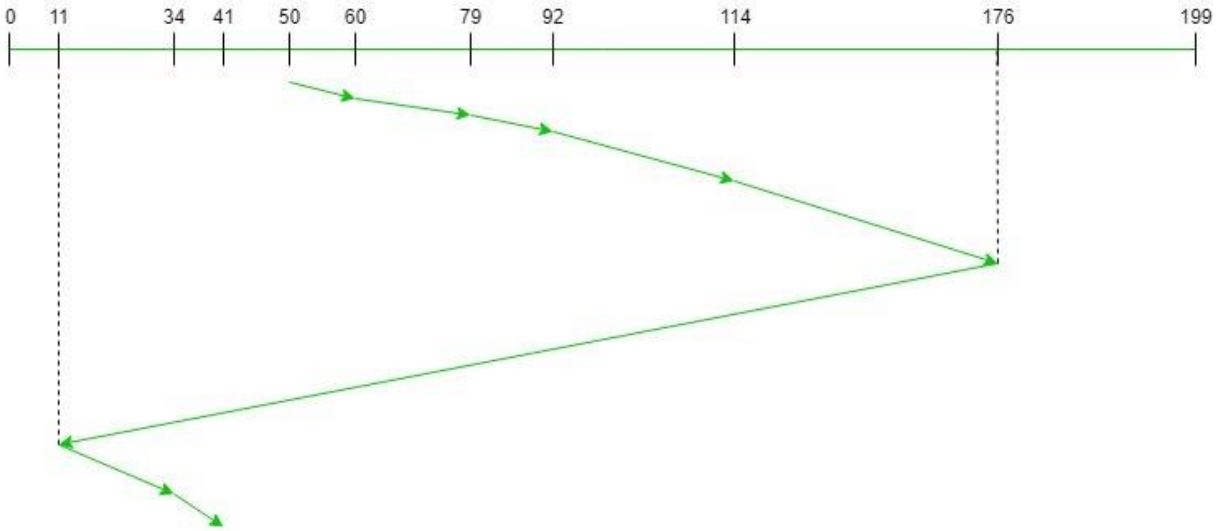


Recognition of the dynamic nature of the request queue leads to the **LOOK algorithm.** The read-write head starts from its present position, services requests while moving towards one end of the disk, after servicing the last request of this end, the direction of the head movement is reversed, and servicing continues as far as the last request in this direction. LOOK scheme continuously scans the disk from one direction to the other keeping the swing to the last request on either side. In a real-time system, if a request arrives just in front of the head, it will be serviced almost immediately, whereas a request arriving just behind the head will have to wait until the head moves to the last request of one end of the disk, reverses direction and returns to service the leftover requests on its way to the other end. In practical systems, the head is moved as far as the last request in each direction. As soon as there are no requests in the current direction, the head movement is reversed.



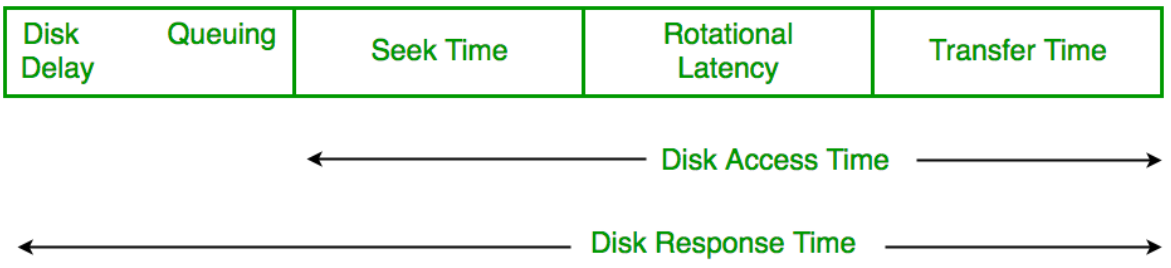
A variant of LOOK scheduling that is designed to provide a more uniform wait time is **C-LOOK (Circular LOOK)** scheduling. CLOOK moves the head towards one direction from its present position,

after servicing the last request in the current direction it reverses its direction and immediately returns to the first request of the other end, without servicing any requests on the return trip. C-LOOK scheduling essentially treats the disk as though it were circular, with the last track adjacent to the first one.



Software Design

Four Different programs have been developed in C++ language for disk scheduling algorithms which includes four major scheduling algorithms (i.e. FCFS, SSTF, LOOK, and C-LOOK). This algorithm runs and presents results based upon service requests and the number of tracks involved in the given test sample. Tracks requests are read by the program from the relevant file. The algorithm automatically offers reordered list of the read requests and a queue is displayed to service the requests. When all requests are serviced, then the result in the form of total head movement is displayed. Each program can be run to get new data of requests from the keyboard. Total head movement is calculated based upon entered data. Various data structures such as linked lists, arrays, queues, and records have been used for successful execution, recording of simulation results.



Implementation

FCFS

```
#include <bits/stdc++.h>
using namespace std;

void FCFS(int arr[], int head,int size){
	int seek_count = 0;
	int distance, cur_track;

	for (int i = 0; i < size; i++) {
		cur_track = arr[i];
		distance = abs(cur_track - head);
		seek_count += distance;
		head = cur_track;
	}
	cout<< "Total number of seek operations = "<<seek_count
<< endl;

	cout << "Seek Sequence is" << endl;

	for (int i = 0; i < size; i++) {
		cout << arr[i] << endl;
	}
}

int main(){
	int n;
	cin>>n;

	int arr[n];
	for(int i=0;i<n;i++){
	    cin>>arr[i];
	}
	int head;
	cin>>head;

	FCFS(arr, head, n);
	return 0;
}
```

SSTF

```
#include <bits/stdc++.h>
using namespace std;
void calculateDifference(int *request, int head, int **diff, int n){
    for(int i = 0; i < n; i++){
        diff[i][0] = abs(head - request[i]);
    }
}

int findMIN(int **diff, int n){
    int index = -1;
    int minimum = 1e9;

    for(int i = 0; i < n; i++){
        if (!diff[i][1] && minimum > diff[i][0]){
            minimum = diff[i][0];
            index = i;
        }
    }
    return index;
}

void shortestSeekTimeFirst(int *request,int head, int n){
    if (n == 0){
        return;
    }


    int **diff = new int*[n];
    for (int i = 0; i < n; ++i) {
        diff[i] = new int [2];
    }
```

```
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < 2; ++j) {
            diff[i][j] = 0;
        }
    }

    int seekCount = 0;

    int *seekSequence = new int[n+1];
    for (int i = 0; i <= n; ++i) {
        seekSequence[i] = 0;
    }

    for(int i = 0; i < n; i++){
        seekSequence[i] = head;
        calculateDifference(request, head, diff, n);
        int index = findMIN(diff, n);
        diff[index][1] = 1;
        seekCount += diff[index][0];
        head = request[index];
    }
    seekSequence[n] = head;

    cout << "Total number of seek operations = " << seekCount << endl;
    cout << "Seek sequence is : " << "\n";

    for(int i = 0; i <= n; i++){
        cout << seekSequence[i] << "\n";
    }
}

int main(){
    int n;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int head;
    cin>>head;
    shortestSeekTimeFirst(arr, head, n);
}
```

```
8
98 183 37 122 14 124 65 67
53
Total number of seek operations = 236
Seek sequence is :
53
65
67
37
14
98
122
124
183
```

FCFS

```
#include <bits/stdc++.h>
using namespace std;
void LOOK(int arr[], int head, string direction,int size){
    int seek_count = 0;
    int distance, cur_track;
    vector<int> downward, upward;
    vector<int> seek_sequence;

    for (int i = 0; i < size; i++) {
        if (arr[i] < head)
            downward.push_back(arr[i]);
        if (arr[i] > head)
            upward.push_back(arr[i]);
    }

    sort(downward.begin(), downward.end());
    sort(upward.begin(), upward.end());

    int run = 2;
    while (run--) {
        if (direction == "downward") {
            for (int i = downward.size() - 1; i >= 0; i--) {
                cur_track = downward[i];
                seek_sequence.push_back(cur_track);
                distance = abs(cur_track - head);
                seek_count += distance;
                head = cur_track;
```

```
            }
            direction = "upward";
        }
        else if (direction == "upward") {
            for (int i : upward) {
                cur_track = i;
                seek_sequence.push_back(cur_track);
                distance = abs(cur_track - head);
                seek_count += distance;
                head = cur_track;
            }
            direction = "downward";
        }
    }

    cout << "Total number of seek operations = "<< seek_count << endl;

    cout << "Seek Sequence is" << endl;
    for (int i : seek_sequence) {
        cout << i << endl;
    }
}

int main(){
    int n;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }

    int head;
    cin>>head;

    //string direction = "upward";
    string direction = "downward";

    LOOK(arr, head, direction,n);
}
```

Upward

```
8
98 183 37 122 14 124 65 67
53
Total number of seek operations = 299
Seek Sequence is
65
67
98
122
124
183
37
14
```

Downward

```
8
98 183 37 122 14 124 65 67
53
Total number of seek operations = 208
Seek Sequence is
37
14
65
67
98
122
124
183
```

CLOOK

```
#include <bits/stdc++.h>
using namespace std;
void CLOOK(int arr[], int head,int size){
    int seek_count = 0;
    int distance, cur_track;

    vector<int> left, right;
```

```
    vector<int> seek_sequence;

    for (int i = 0; i < size; i++) {
        if (arr[i] < head)
            left.push_back(arr[i]);
        if (arr[i] > head)
            right.push_back(arr[i]);
    }

    sort(left.begin(), left.end());
    sort(right.begin(), right.end());

    for (int i : right) {
        cur_track = i;
        seek_sequence.push_back(cur_track);
        distance = abs(cur_track - head);
        seek_count += distance;
        head = cur_track;
    }

    seek_count += abs(head - left[0]);
    head = left[0];
    for (int i : left) {
        cur_track = i;
        seek_sequence.push_back(cur_track);
        distance = abs(cur_track - head);
        seek_count += distance;
        head = cur_track;
    }

    cout << "Total number of seek operations = "<< seek_count << endl;

    cout << "Seek Sequence is" << endl;

    for (int i : seek_sequence) {
        cout << i << endl;
    }
}

int main(){
    int n;
    cin>>n;
```

```
    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int head;
    cin>>head;

    CLOOK(arr, head,n);
}
```

```
8
98 183 37 122 14 124 65 67
53
Total number of seek operations = 322
Seek Sequence is
65
67
98
122
124
183
14
37
```

Testing

Five test samples were selected such that each sample consists of different types and numbers of tracks. The number of tracks varies from 8 to 50. Each sample is executed using a selected algorithm to obtain total head movements.

Test sample 1:
It consists of **8** tracks **[98, 183, 37, 122, 14,124, 65, and 67]**, where track 98 contains the first received request and track 67 has the last received request.

The initial head position was supposed to be at track 53.

When this sample is executed the total head movement for **FCFS is 640 tracks, for SSTF is 236 tracks, and for C-LOOK is 322 tracks**. In case of LOOK there are two possibilities (i.e. either the head can first travel upward or downward). Total head movement for upward movement is **299** tracks and for downward movement are **208** tracks.

Test sample 2:
It contains **12** tracks: **[2, 5, 7, 10, 15, 17, 19, 23, 26, 31 and 35**]. Track 2 is the first request and track 35 is the last request.
Test sample 2 is unique in the sense that all requests are situated below **The initial head position is 53** as well as all requests received are in the ascending order. When this test sample is executed, the total head movement obtained for **FCFS is 84 tracks, for SSTF is 51 tracks and for C-LOOK is 84 tracks**. In case of LOOK algorithm for this example, only downward movement is considered as all requests are situated below the initial head position. So the total head movement in case of **LOOK is 51 tracks**.

Test Sample 3:
There are **24** tracks: **[47, 35, 180, 132, 156, 23, 34, 5, 210, 83, 96, 103, 88, 76, 113, 128, 73, 120, 169, 123, 111, 179, 136 and 40]**.
Track 47 is the first and track 40 is the last request.
The initial head position at track 53.
On its execution, the total head movement obtained for **FCFS is 1255 tracks, for SSTF are 253 tracks and for C-LOOK are 404 tracks**.
LOOK gives the total head movement of **362 tracks and 253 tracks** for upward and downward head movement respectively.

Test Sample 4:
There are **24** tracks: **[77, 5, 11, 175, 25, 38, 1, 20, 250, 17,211, 15, 37,285, 149, 42, 19, 32, 2, 31, 6, 34, 48 and 125]**.
Track 77 is the first and track 125 is the last request.
The initial position of head is at track 53,
The total head movement for **FCFS is 2290 tracks, for SSTF is 336 tracks, C-LOOK is 563 tracks, for upward movement in LOOK is 516 and for downward movement is 336**.

Test sample 5:
It consists of **50** tracks: **[272, 69, 23, 58, 190, 205, 39, 117, 25, 213, 121, 290, 310, 19, 80, 8, 94, 222, 40, 176, 312, 303, 260, 150, 167, 32, 27, 253, 122, 73, 134, 11, 5, 83, 50, 2, 240, 240, 301, 4, 263, 138, 7, 291, 319, 9, 15, 230, 88 and 1]**.
The initial position of head is at track 53.

The total head movement obtained for **FCFS is 5804 tracks, for SSTF is 590 tracks, for C-LOOK is 633 Tracks**, for **LOOK upward movement is 584 tracks** and for **LOOK for downward movement is 370 tracks.**

S/No	Total Tracks	Algorithm	Initial Head Position	Head Movement Direction	Total Head Movement (in tracks)
1.	8	FCFS	53	N.A	640
2.	8	SSTF	53	N A	236
3.	8	LOOK	53	U/WARD	299
4.	8	LOOK	53	D/WARD	208
5.	8	C-LOOK	53	N.A	322
6.	12	FCFS	53	N.A	84
7.	12	SSTF	53	N A	51
8.	12	LOOK	53	D/WARD	51
9.	12	C-LOOK	53	N.A	84
10.	24	FCFS	53	N.A	1255
11.	24	SSTF	53	N A	253
12.	24	LOOK	53	U/WARD	362
13.	24	LOOK	53	D/WARD	253
14.	24	C-LOOK	53	N.A	404
15.	24	FCFS	53	N.A	2290
16.	24	SSTF	53	N A	336
17.	24	LOOK	53	U/WARD	516
18.	24	LOOK	53	D/WARD	336
19.	24	C-LOOK	53	N.A	563
20.	50	FCFS	53	N.A	5804
21.	50	SSTF	53	N A	590
22.	50	LOOK	53	U/WARD	584
23.	50	LOOK	53	D/WARD	370
24.	50	C-LOOK	53	N.A	633

Conclusion

The actual performance of disk I/O depends on computer architecture, operating system, nature of the I/O channel, disk structure ,and disk scheduling algorithms. On a movable-head system, **total delay in an I/O operation contains seek, latency and transfer time.** A Major portion of delay is caused by the seek time and it must be minimized in order to enhance the performance of disk scheduling. An efficient disk service requires that the head be moved as quickly as possible to the required track to perform a read/write operation. A comprising of four disk scheduling algorithms **(FCFS, SSTF, LOOK, and C-LOOK)** has been implemented in C++ language to reduce the seek time. Each of these algorithms is used for selecting a request for servicing from the queue of requests. The queue of requests is maintained in an appropriate order based upon the scheduling algorithm. Five test samples containing requests for varying tracks (8 to 50) were used to check the performance

of the simulator using four algorithms. **The Initial head position for each case was track 53.** Head movements of each algorithm have been illustrated using test samples Tabulated simulation results indicated that **FCFS provides the worst performance due to the presence of wild swings**. FCFS goes for a lengthy seek to service a distant waiting request even though another request may have just arrived near the track where the read/write head is currently positioned. Its total head movement is 640, 84, 1255, 2290, and 5804 tracks for test samples 1, 2, 3, 4, and 5 respectively. **In the SSTF algorithm, a great improvement has been observed** due to the elimination of large swings. It reduces the head movements and provides results of 236, 51, 253, 336, and 90 tracks for five test samples. LOOK algorithm works like SSTF except that it uses either upward or downward direction to choose the requests from the rearranged queue. Performance of this algorithm was 299 (sample l), 362 (sample 3), 516 (sample 4), and 584 (sample 5) using upward direction whereas it was 208 (sample l), 51(sample 2), 253 (sample 3), 336 (sample 4), and 370 (sample 5) using downward direction. Thus, **it demonstrated the best results for head movement in the downward direction and very good results for head movement in the upward direction**. Results obtained for C-LOOK were 322, 84, 404, 563, and 633 for test samples 1 to 5. **Its performance remains worse than LOOK and SSTF in most of the cases and provides better results than FCFS**. **CLOOK is found more efficient at high loads only**. These results are extremely useful for designing more efficient and effective disk scheduling algorithms to reduce seek time in multiprogramming environments.