# A Project Report

# On

# Comparative Analysis of Disk Scheduling Algorithms

# For

## Operating Systems and System Programming Lab (15B17CI472)



**Submitted by:**

| | |
|---|---|
| Anya Rathi | 9920103001 |
| Devansh Chugh | 9920103011 |
| Vaishali Ranjan | 9920103013 |
| Aviral Gupta | 9920103021 |
| Khushi Kalra | 9920103025 |

**Submitted to:**

Dr. Anubhuti Roda Mohindra

Dr. Gaurav Kumar Nigam

Dr. Neeraj Jain

Dr. Charu

**Department of CSE/IT**

**Jaypee Institute of Information Technology University, Noida**


**December, 2022**

## Chapter 1: Problem Statement

A process needs two types of time, CPU time and I/O time. For I/O, it requests the Operating system to access the disk. However, the operating system must be fair enough to satisfy each request and at the same time, theoperating system must maintain the efficiency and speed of process execution. So, it must determine which request should be satisfied next.

Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

## Chapter 2: Introduction

The file system can be viewed logically in three different divisions i.e. user, programmer interface to the file system and secondary storage structure. The lowest level of the file system is secondary storage structure and disk is the main secondary storage device thatis generally divided into tracks, cylinders and sectors and stores the data permanently. The I/O operation depends on the computer system, the operating system, and the natureof the I/O channel and disk controller hardware.

The user programs make use of the data on the disk by means of I/O requests. Data is stored on both surfaces of a series of magnetic disks called platters that are connected by a single spindle. The surface of a platter is logically divided into tracks that are further subdivided into sectors and the set of tracks that are at one arm position form a cylinder.

One read-write head per disk surface is used to access the data and all read-write headsare attached to a single moving arm. The segment of the disk surface where the data isread or written must revolve under the read-write head for accessing the data. The key responsibility of the operating system is to efficiently use the hardware of the computer system. For most disks, the seek time leads to latency time and transfer time, so reducing the mean seek time can improve system performance to a large extent.

## 2.1 Motivation

Disk scheduling is a policy of the operating system to decide which I/O request is going to be satisfied foremost. The goal of disk scheduling algorithms is to maximize the throughput and minimize the response time. The present piece of investigation documents the comparative analysis of six different disk scheduling algorithms viz. First Come First Serve (FCFS), Shortest Seek Time First (SSTF), Scan, C-Scan, Look and C-look disk scheduling by comparing their head movement in different runs. The implementation is carried out by creating an interface tocalculate total head movement of these six algorithms.

## 2.2 Objective

In multiprogramming systems, processes running concurrently may generate requests for reading and writing disk records. The operating system handles these I/O requests from the queue and processes them one by one. The algorithm used to choose which I/Orequest is going to be fulfilled earliest is called disk scheduling algorithm. The different disk scheduling algorithms are First Come First Serve, Shortest Seek Time First, Scan, Look, Circular Scan and Circular Look. The main objectives for any disk scheduling algorithm are minimizing the response time and maximizing the throughput.

## 2.3 Contribution

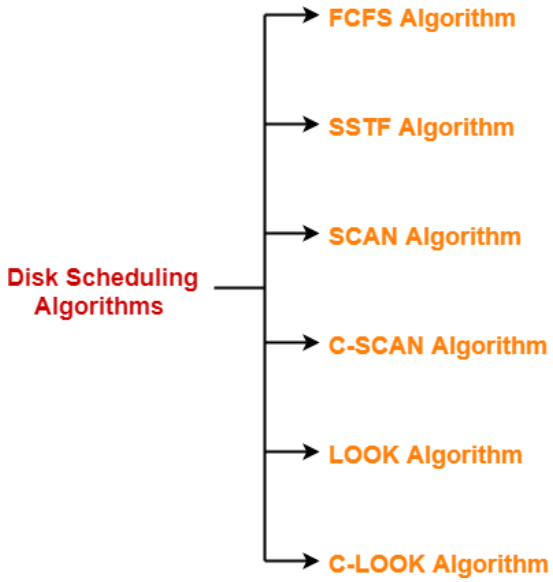
| ANYA (9920103001) | DEVANSH (9920103011) | VAISHALI (9920103013) | AVIRAL (9920103021) | KHUSHI (9920103025) |
|---|---|---|---|---|
| Implementation of SCAN algorithm. | Implementation of FCFS algorithm and the main function. | Implementation of LOOK algorithm. | Implementation of SSTF algorithm. | Implementation of C-SCAN algorithm. |

C-LOOK Algorithm, findMIN and calculatedifference functions have been implemented by all the group members.

# Chapter 3: Detailed description of the project

## 3.1 Working Procedure of Disk Scheduling Algorithms

Disk scheduling algorithms are applied to decide which I/O request is going to be satisfied first. The fundamental disk scheduling algorithms are First Come First Serve, Shortest Seek Time First, Scan, Look, Circular Scan and Circular Look.




## 3.2 First Come First Serve Disk Scheduling Algorithm (FCFS)

The functioning of this algorithm is maintained by the First in First out (FIFO) queue. With this scheme, the I/O requests are served or processed according to their arrival. Though this algorithm improves response time but fails to decrease the average seek timebecause this algorithm needs a lot of random head movements and disk rotations.

### 3.3 Shortest Seek Time First Disk Scheduling Algorithm (SSTF)

In this approach, the read/write head serves the request first that has minimum seek timefrom the current head position. The I/O requests that are close to the read/write head areserviced first. SSTF disk scheduling algorithm is actually a form of SJF scheduling algorithm and may cause starvation of some requests. SSTF disk scheduling algorithm has better throughput than FCFS disk scheduling algorithm but some requests may be delayed for a long time if lots of closely situated requests arrive just after it.

### 3.4 Scan Disk Scheduling Algorithm (SCAN)

In the Scan disk scheduling algorithm, the read/write head starts from one end and movestowards the other end and servicing requests as it reaches each track until it reaches theother end of the disk. The direction of the read/write head reverses after reaching at the other end and servicing continues. In this way, the read/write head continuously swings from end to end**.**

### 3.5 Look Disk Scheduling Algorithm (LOOK)

Look disk scheduling algorithm is the improved version of Scan disk scheduling algorithmand avoids the starvation problem. In Look disk scheduling algorithm, the arm goes onlyas far as final requests in each direction and then reverses direction without going all the way to the end . In this way it improves both response time and throughput.

## 3.6 Circular Scan Disk Scheduling Algorithm (C-SCAN)

Cyclic Scan or Circular Scan disk scheduling algorithm is an improved version of Scan disk scheduling algorithm and known as one directional scan. It starts its scan towards the nearest end and services the requests all the way to the end.

## 3.7 Comparison of different scheduling algorithms

Table-1 compares the average head movement of six disk scheduling algorithms for thefirst five runs and their average. Similar requests are assigned for every individual run for all six algorithms and their total head movement is calculated.

| S.No. | FCFS | SSTF | Scan | Look | C-Scan | C-Look |
|---|---|---|---|---|---|---|
| 1 | 540 | 240 | 290 | 240 | 363 | 295 |
| 2 | 631 | 276 | 280 | 276 | 348 | 306 |
| 3 | 264 | 217 | 270 | 224 | 393 | 289 |
| 4 | 322 | 189 | 235 | 189 | 363 | 189 |
| 5 | 640 | 235 | 275 | 245 | 378 | 300 |
| **Average** | **479** | **231** | **270** | **235** | **369** | **276** |

*Table 1: Average head movement of first five runs and their average.*

The Shortest Seek Time First algorithm produces the minimum head movement of 240 inthe first run, 276 in the second run, 217 in the third run, 189 in the fourth run, 235 in the last run and average head movement of all the runs is 231. From the above table, it is verymuch clear that for the different five runs, the

Shortest Seek Time First disk scheduling algorithm has minimum total head movement in all cases as compared to other five algorithms. A graphical representation is shown in the figure 3.1 with the help of data evaluated in table-1.

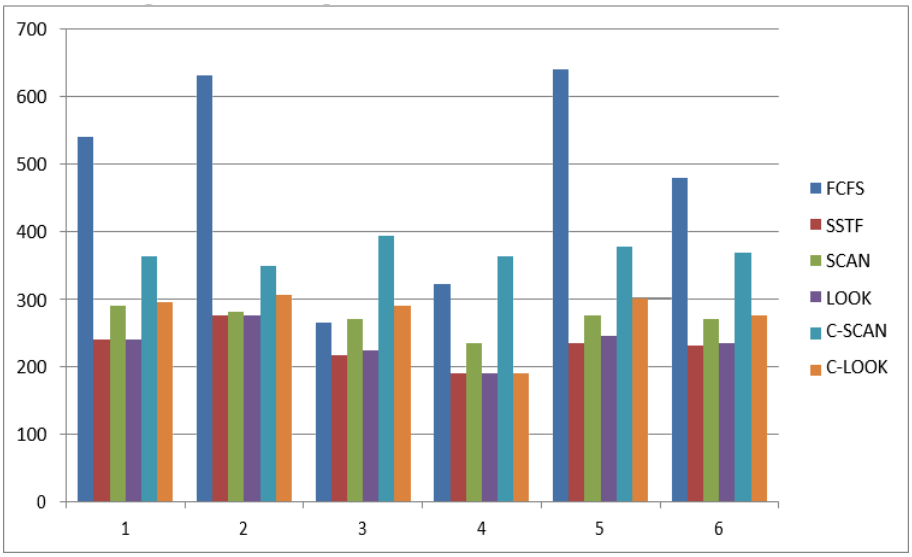



*Figure 3.1: Comparison between six disk scheduling algorithms in graphical representation.*

The figure 3.1 gives comparative details of the six disk scheduling algorithms. The X axis represents five runs and the average case and the Y axis is used for calculating averagetotal head movement of each algorithm. From the various runs, it is concluded thatShortest Seek Time First has the minimum average head movement than all other five algorithms for the similar requests. Thus the Shortest Seek Time First algorithm is a good criterion for selecting the requests for I/O from the disk queue.

# Chapter 4: Implementation

## 4.1 Workflow Diagram

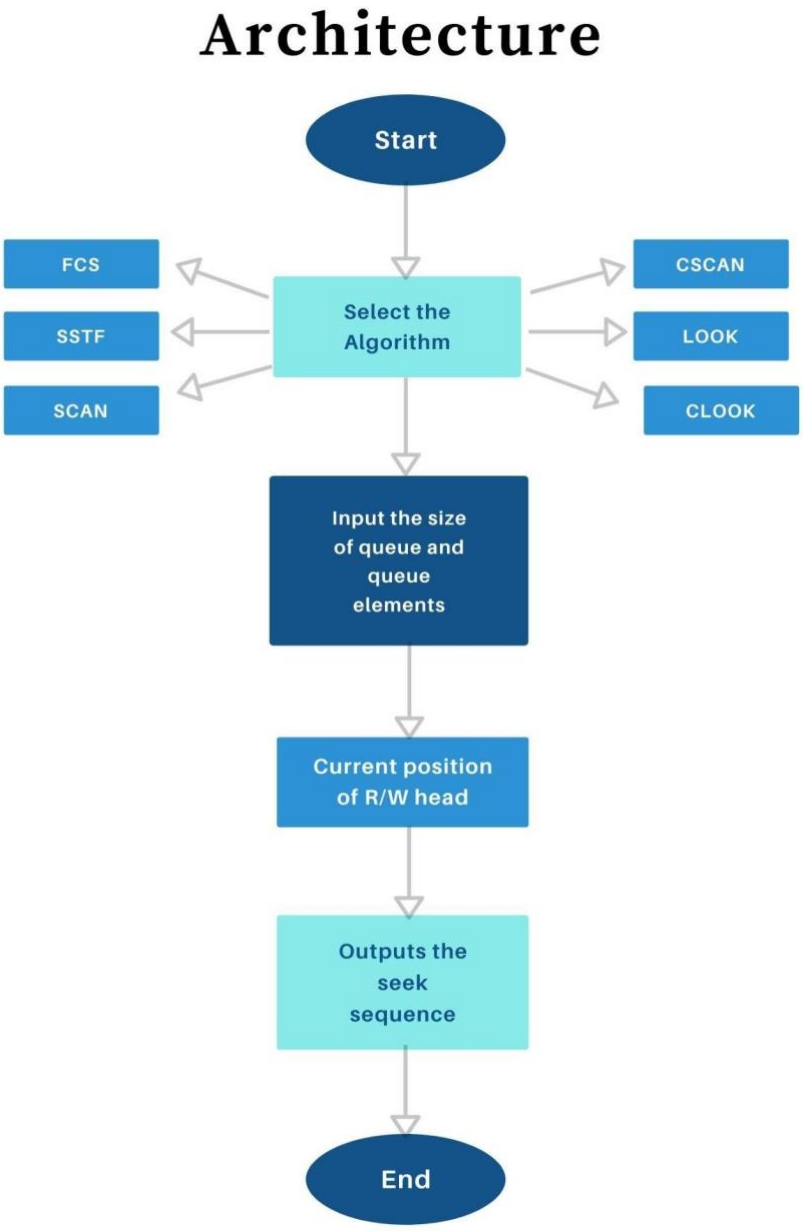



*Figure 4.1: Flow diagram*

## 4.2 Program Code

### MAIN FUNCTION

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

#define HIGH 199
#define LOW 0

void fcfs();
void sstf();
void scan();
void cscan();
void look();
void clook();

int main(){

  printf("Enter 1 for FCFS \nEnter 2 for SSTF \nEnter 3 for SCAN \nEnter 4 for CSCAN \nEnter 5 for LOOK \nEnter 6 for CLOOK \n");
  int choice;
  scanf("%d",&choice);
  switch(choice)
  {
      case 1:
          fcfs();
      break;

      case 2:
          sstf();
      break;

      case 3:
          scan();
      break;

      case 4:
          cscan();
      break;

      case 5:
       look();
      break;

      case 6:
        clook();
      break;


       default:
          printf("You fool, don't know english? Get Lost");
    }
  return 0;
}
```

### FCFS

```
void fcfs()
{
        int queue[100], q_size, head, seek =0, diff;
        float avg;

        printf("%s\n", "**FCFS Disk Scheduling Algorithm**");

        printf("%s\n", "Enter the size of the queue");
        scanf("%d", &q_size);

        printf("%s\n", "Enter queue elements");
        for(int i=1; i<=q_size; i++){
        scanf("%d",&queue[i]);
        }

        printf("%s\n","Enter initial head position");
        scanf("%d", &head);
            queue[0]=head;

        for(int j=0; j<=q_size-1; j++){
            diff = abs(queue[j]-queue[j+1]);
            seek += diff;
            printf("Move from %d to %d with Seek %d\n",queue[j],queue[j+1],diff);
            }

        printf("\nTotal seek time is %d\t",seek);
        avg = seek/(float)q_size;
        printf("\nAverage seek time is %f\t", avg);
}
```

```
void calculatedifference(int request[], int head,
                         int diff[][2], int n)
{
    for(int i = 0; i < n; i++)
    {
        diff[i][0] = abs(head - request[i]);
    }
}

// Find unaccessed track which is
// at minimum distance from head
int findMIN(int diff[][2], 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;
}
```

## SSTF

```
void sstf()
{
    int n;
    printf("%s\n", "Enter the size of the queue");
    scanf("%d", &n);
    int request[n];
    printf("%s\n", "Enter queue elements");
    for(int i=0; i<n;i++)
    {
        scanf("%d",&request[i]);
    }

    int head;
    printf("%s\n","Enter initial head position");
    scanf("%d", &head);


    if (n == 0)
    {
        return;
    }

    // Create array of objects of class node
    int diff[n][2] = { { 0, 0 } };
```

```
    // Count total number of seek operation
    int seekcount = 0;

    // Stores sequence in which disk access is done
    int seeksequence[n + 1] = {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;

        // Increase the total count
        seekcount += diff[index][0];

        // Accessed track is now new head
        head = request[index];
    }
    seeksequence[n] = head;

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

    // Print the sequence
    for(int i = 0; i <= n; i++)
    {
        cout << seeksequence[i] << " -> ";
    }
}
```

## SCAN

```
void scan()
{

    int size;
    printf("%s\n", "Enter the size of the queue");
    scanf("%d", &size);
    int arr[size];
    printf("%s\n", "Enter queue elements");
    for(int i=0; i<size;i++)
    {
        scanf("%d",&arr[i]);
    }

    int head,direction;
    printf("%s\n","Enter initial head position");
    scanf("%d", &head);

    printf("%s\t", "Enter the direction 0-left, 1-right");
    scanf("%d", &direction);

    //string direction = "left";
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;

    // appending end values
    // which has to be visited
    // before reversing the direction
    if (direction == 0)
        left.push_back(0);
    else if (direction == 1)
        right.push_back(200 - 1);

    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]);
    }
```

```
    // sorting left and right vectors
    std::sort(left.begin(), left.end());
    std::sort(right.begin(), right.end());

    // run the while loop two times.
    // one by one scanning right
    // and left of the head
    int run = 2;
    while (run--) {
        if (direction == 0) {
            for (int i = left.size() - 1; i >= 0; i--) {
                cur_track = left[i];

                // appending current track to seek sequence
                seek_sequence.push_back(cur_track);

                // calculate absolute distance
                distance = abs(cur_track - head);

                // increase the total count
                seek_count += distance;

                // accessed track is now the new head
                head = cur_track;
            }
            direction = 1;
        }
        else if (direction == 1) {
            for (int i = 0; i < right.size(); i++) {
                cur_track = right[i];
                // appending current track to seek sequence
                seek_sequence.push_back(cur_track);

                // calculate absolute distance
                distance = abs(cur_track - head);

                // increase the total count
                seek_count += distance;

                // accessed track is now new head
                head = cur_track;
            }
            direction = 0;
        }
    }
```

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

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

    for (int i = 0; i < seek_sequence.size(); i++)
        cout << seek_sequence[i] <<" -> ";
    }
}
```

# CSCAN

```c
void cscan()
{
    int queue[20], q_size, head, i,j, seek=0, diff, max, temp, queue1[20],
  float avg;

  printf("%s\t", "Input no of disk locations");
  scanf("%d", &q_size);

  printf("%s\t", "Enter initial head position");
  scanf("%d", &head);

  printf("%s\n","Enter disk positions to be read");
  for(i=0; i<q_size; i++) {

    scanf("%d", &temp);
     if(temp >= head){
        queue1[temp1] = temp;
        temp1++;
     } else {
        queue2[temp2] = temp;
        temp2++;
     }
  }
  //sort both arrays
  for(i=0; i<temp1-1; i++){
    for(j=i+1; j<temp1; j++){
       if(queue1[i] > queue1[j]){
        temp = queue1[i];
        queue1[i] = queue1[j];
        queue1[j] = temp;
       }
    }
  }

  for(i=0; i<temp2-1; i++){
    for(j=i+1; j<temp2; j++){
     if(queue2[i]>queue2[j]){
        temp = queue2[i];
        queue2[i] = queue2[j];
        queue2[j] = temp;
     }
    }
  }
```

```c
  //calculate closest edge
  if(abs(head-LOW) >= abs(head-HIGH)){

     for(i=1,j=0; j<temp1; i++,j++){
        queue[i] = queue1[j];
     }

        queue[i] = HIGH;
        queue[i+1] = 0;

     for(i=temp1+3, j=0; j<temp2; i++, j++){
        queue[i] = queue2[j];
     }

  } else {

     for(i=1,j=temp2-1; j>=0; i++,j--){
        queue[i] = queue2[j];
     }

        queue[i] = LOW;
        queue[i+1] = HIGH;

     for(i=temp2+3, j=temp1-1; j>=0; i++, j--){
        queue[i] = queue1[j];
     }
}

  queue[0] = head;

  for(j=0; j<=q_size+1; j++){
     diff=abs(queue[j+1] - queue[j]);
       seek += diff;
       printf("Disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);

  }

  //seek = seek - max; //subtract seek time back to zero
  printf("Total seek time is %d\n", seek);
  avg = seek/(float)q_size;
  printf("Average seek time is %f\n", avg);
}
```

# LOOK

```c
void look()
{
    int queue[20], head, q_size, i,j, seek=0, diff, max, temp, queue1[20], queue2[
  float avg;
int dir;
  printf("%s\t", "Input the number of disk locations");
  scanf("%d", &q_size);

  printf("%s\t", "Enter initial head position");
  scanf("%d", &head);

  printf("%s\t", "Enter the direction 0-left, 1-right");
  scanf("%d", &dir);

  printf("%s\n", "Enter disk positions to read");

  for(i=0; i<q_size; i++){
    scanf("%d", &temp);
    //queue1 - elems greater than head
    if(temp >= head){
       queue1[temp1] = temp;
       temp1++;
    } else {
       queue2[temp2] = temp;
       temp2++;
    }
  }

  //sort queue1 - increasing order
  for(i=0; i<temp1-1; i++){
   for(j=i+1; j<temp1; j++){
      if(queue1[i] > queue1[j]){
       temp = queue1[i];
       queue1[i] = queue1[j];
       queue1[j] = temp;
      }
    }
  }

  //sort queue2 - decreasing order
  for(i=0; i<temp2-1; i++){
    for(j=i+1; j<temp2; j++){
      if(queue2[i] < queue2[j]){
         temp = queue2[i];
         queue2[i] = queue2[j];
         queue2[j] = temp;
```

```c
      }
    }
  }

  if(dir==1){

     for(i=1,j=0; j<temp1; i++,j++){
        queue[i] = queue1[j];
     }

     for(i=temp1+1, j=0; j<temp2; i++, j++){
        queue[i] = queue2[j];
     }

  } else {

     for(i=1,j=0; j<temp2; i++,j++){
        queue[i] = queue2[j];
     }

     for(i=temp2+1, j=0; j<temp1; i++, j++){
        queue[i] = queue1[j];
     }

  }

  queue[0] = head;

  for(j=0; j<q_size; j++){
     diff=abs(queue[j+1] - queue[j]);
       seek += diff;
       printf("Disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);

  }

  printf("Total seek time is %d\n", seek);
  avg = seek/(float)q_size;
  printf("Average seek time is %f\n", avg);
}
```

# C-LOOK

```
void clook()
{
    int size;
    printf("%s\n", "Enter the size of the queue");
    scanf("%d", &size);
    int arr[size];
    printf("%s\n", "Enter queue elements");
    for(int i=0; i<size;i++)
    {
        scanf("%d",&arr[i]);
    }

    int head;
    printf("%s\n","Enter initial head position");
    scanf("%d", &head);

    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;

    // Tracks on the left of the
    // head will be serviced when
    // once the head comes back
    // to the beginning (left end)
    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]);
    }

    // Sorting left and right vectors
    std::sort(left.begin(), left.end());
    std::sort(right.begin(), right.end());

    // First service the requests
    // on the right side of the
    // head
    for (int i = 0; i < right.size(); i++) {
        cur_track = right[i];

        // Appending current track to seek sequence
        seek_sequence.push_back(cur_track);

        // Calculate absolute distance
        distance = abs(cur_track - head);

        // Increase the total count
        seek_count += distance;

        // Accessed track is now new head
        head = cur_track;
    }

    // Once reached the right end
    // jump to the last track that
    // is needed to be serviced in
    // left direction
    seek_count += abs(head - left[0]);
    head = left[0];

    // Now service the requests again
    // which are left
    for (int i = 0; i < left.size(); i++) {
        cur_track = left[i];

        // Appending current track to seek sequence
        seek_sequence.push_back(cur_track);

        // Calculate absolute distance
        distance = abs(cur_track - head);

        // Increase the total count
        seek_count += distance;

        // Accessed track is now the new head
        head = cur_track;
    }

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

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

    for (int i = 0; i < seek_sequence.size(); i++) {
        cout << seek_sequence[i] <<" -> ";
    }
}
```

# Chapter 5: Results

We have calculated the average total head movement after entering the various runs forthe requests of different algorithms because total head movement is the criteria for analyzing the disk scheduling algorithms. After comparing the total head movement of various algorithms, we have found that the Shortest Seek Time First disk scheduling algorithm has the least average head movement than the others discussed above in context to total head movement.

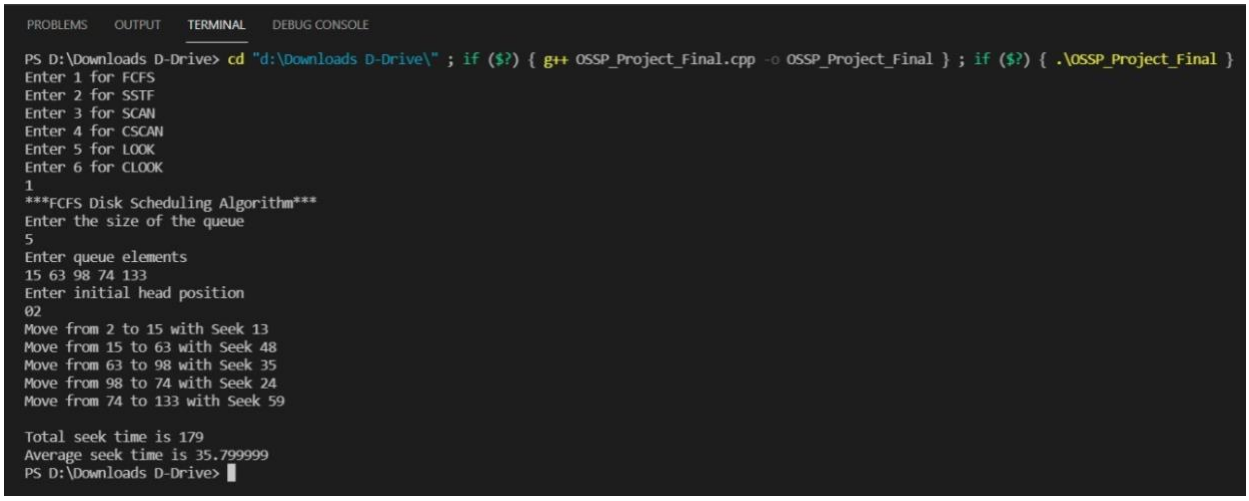



*Figure 4.2: FCFS*

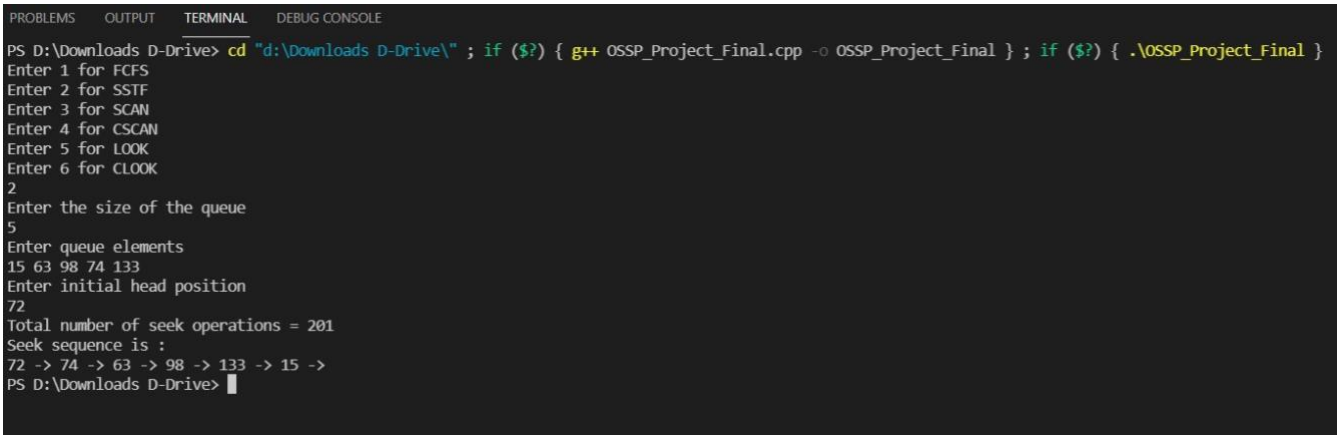



*Figure 4.3: SSTF*

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\Downloads D-Drive> cd "d:\Downloads D-Drive\" ; if ($?) { g++ OSSP_Project_Final.cpp -o OSSP_Project_Final } ; if ($?) { .\OSSP_Project_Final }
Enter 1 for FCFS
Enter 2 for SSTF
Enter 3 for SCAN
Enter 4 for CSCAN
Enter 5 for LOOK
Enter 6 for CLOOK
3
Enter the size of the queue
5
Enter queue elements
15 63 98 74 133
Enter initial head position
100
Total number of seek operations = 233
Seek Sequence is
98 -> 74 -> 63 -> 15 -> 0 -> 133 ->
PS D:\Downloads D-Drive>
```

*Figure 4.4: SCAN*

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\Downloads D-Drive> cd "d:\Downloads D-Drive\" ; if ($?) { g++ OSSP_Project_Final.cpp -o OSSP_Project_Final } ; if ($?) { .\OSSP_Project_Final }
Enter 1 for FCFS
Enter 2 for SSTF
Enter 3 for SCAN
Enter 4 for CSCAN
Enter 5 for LOOK
Enter 6 for CLOOK
4
Input no of disk locations      5
Enter initial head position     02
Enter disk positions to be read
15 63 98 74 133
Disk head moves from 2 to 0 with seek 2
Disk head moves from 0 to 199 with seek 199
Disk head moves from 199 to 133 with seek 66
Disk head moves from 133 to 98 with seek 35
Disk head moves from 98 to 74 with seek 24
Disk head moves from 74 to 63 with seek 11
Disk head moves from 63 to 15 with seek 48
Total seek time is 385
Average seek time is 77.000000
PS D:\Downloads D-Drive>
```

*Figure 4.5: CSCAN*

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\Downloads D-Drive> cd "d:\Downloads D-Drive\" ; if ($?) { g++ OSSP_Project_Final.cpp -o OSSP_Project_Final } ; if ($?) { .\OSSP_Project_Final }
Enter 1 for FCFS
Enter 2 for SSTF
Enter 3 for SCAN
Enter 4 for CSCAN
Enter 5 for LOOK
Enter 6 for CLOOK
5
Input the number of disk locations      5
Enter initial head position     88
Enter disk positions to read
15 63 98 74 133
Disk head moves from 88 to 74 with seek 14
Disk head moves from 74 to 63 with seek 11
Disk head moves from 63 to 15 with seek 48
Disk head moves from 15 to 98 with seek 83
Disk head moves from 98 to 133 with seek 35
Total seek time is 191
Average seek time is 38.200001
PS D:\Downloads D-Drive>
```

*Figure 4.6: LOOK*


*Figure 4.7: CLOOK*

## Chapter 6: Conclusion

The report analyzes the six disk scheduling algorithms viz. First Come First Serve, Shortest Seek Time First, Scan, Look, C-Scan and C-Look. Different disk scheduling algorithms can be used depending upon the load of the system. The performance depends upon the number and types of requests.

# Chapter 7: References


[1] https://www.geeksforgeeks.org/disk-scheduling-algorithms/

[2] https://www.gatevidyalay.com/disk-scheduling-disk-scheduling-algorithms/

[3] http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html

[4] https://www.javatpoint.com/os-disk-scheduling

[5] https://solver.assistedcoding.eu/disk_scheduling