**“TO STIMULATE THE WORKING OF FCFS(FIRST COME FIRST SERVE), SSTF(SHORTEST SEEK TIME FIRST), SCAN, C-SCAN, LOOK AND C-SCAN DISC SCHEDULING ALGORITHMS”**

**at**

**CSE, MODEL INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**BACHELOR OF TECHNOLOGY (Computer Engineering)**

**SUBMITTED BY-** Japleen Kaur 2021A1R033

Gokul Jamwal 2021A1R043

Dhruv Gupta 2021A1R041

Reetikesh Bali 2021A1R037

**BRANCH -** CSE

**SEMESTER-** 3rd

**EMAIL ID-** [2021a1r033@mietjammu.in](mailto:2021a1r033@mietjammu.in)

[2021a1r043@mietjammu.in](mailto:2021a1r043@mietjammu.in)

[2021a1r041@mietjammu.in](mailto:2021a1r041@mietjammu.in)

[2021a1r037@mietjammu.in](mailto:2021a1r037@mietjammu.in)

**ACKNOWLEDGEMENT**

We take this opportunity to express our sincere gratitude to all those who helped us in various capacities in undertaking this project and devising the report.

We are privileged to express our sense of gratitude to our faculty guide Asst. Professor Saurabh Sharma whose unparalleled knowledge, moral fiber and judgment along with his know-how , was an immense support in completing this project.

We are also grateful to Dr. Ashok Kumar, Dean Academics for the brainwave and encouragement given.

We are also entitled to Professor Ankur Gupta, the director of MIET, for their consistent support and motivation which leads us to a better future.

We take this opportunity to thank our friends for their cooperation and compliance.

**TABLE OF CONTENTS**

**HEADING PAGE NO:**

**1.** Project Title

**2.** Acknowledgement

**3.** Table Of Contents

**4.** Project Summary

I. Introduction 1

II. Objectives 2

III. Methodology ( flowchart ) 3

IV. Algorithm 4

V. Implementation 9

VI. Test ( outputs ) 23

VII. References 27

**PROBLEM STATEMENT 9**

**“TO STIMULATE THE WORKING OF FCFS(FIRST COME FIRST SERVE), SSTF(SHORTEST SEEK TIME FIRST), SCAN, C-SCAN, LOOK AND C-SCAN DISC SCHEDULING ALGORITHMS”**

**ABSTRACT**

Disk scheduling is a policy of the operating system to decide which I/O request is going to be satisfied first. 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, Shortest Seek Time First, Scan, Look, C-Scan and C-look disk scheduling by comparing their head movement in different runs. The implementation is carried out in Turbo C by creating an interface to calculate total head movement of these six algorithms. I 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 that is 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 nature of 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 heads are.

**OBJECTIVES**

The main purpose of the disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

### Goal of Disk Scheduling Algorithm

* Fairness
* High throughout
* Minimal traveling head t

**METHODOLOGY (FLOWCHART)**

Diagram

Description automatically generatedDiagram

Description automatically generated

**ALGORITHM**

1. **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.

1. 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.

2. Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.

3. Increment the total seek count with this distance.

4. Currently serviced track position now becomes the new head position

5. Go to step 2 until all tracks in the request array have not been serviced.

1. **SSTF**

Basic idea is the tracks which are closer to the current disk head position should be serviced first in order to minimize the seek operations.

1. Let Request array represents an array storing indexes of tracks that have been requested. ‘head’ is the position of disk head.

2. Find the positive distance of all tracks in the request array from head.

3. Find a track from the requested array which has not been accessed/serviced yet and has minimum distance from head.

4. Increment the total seek count with this distance.

5. Currently serviced track position now becomes the new head position.

6. Go to step 2 until all tracks in the request array have not been serviced.

1. **LOOK**

LOOK is the advanced version of SCAN (elevator) disk scheduling algorithm which gives slightly better seek time. TheLOOK 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 starts 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.

1. 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.

2. The initial direction in which the head is moving is given and it serves in the same direction.

3. The head services all the requests one by one in the direction the head is moving.

4. The head continues to move in the same direction until all the request in this direction are

not finished.

5. While moving in this direction calculate the absolute distance of the track from the head.

6. Increment the total seek count with this distance.

7. Currently serviced track position now becomes the new head position.

8. Go to step 5 until we reach the last request in this direction.

9. 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 the request array have not been serviced.

**4. C-LOOK**

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.

1. 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.

2. The initial direction in which the head is moving is given and it services in the same direction.

3. The head services all the requests one by one in the direction it is moving.

4. The head continues to move in the same direction until all the requests in this direction have been serviced.

5. While moving in this direction, calculate the absolute distance of the tracks from the head.

6. Increment the total seek count with this distance.

7. Currently serviced track position now becomes the new head position.

8. Go to step 5 until we reach the last request in this direction.

9. 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.

10. Reverse the direction and go to step 3 until all the requests have not been serviced.

**5. SCAN**

In SCAN 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.

1. 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.

2. Let direction represents whether the head is moving towards left or right.

3. In the direction in which the head is moving, service all tracks one by one.

4. Calculate the absolute distance of the track from the head.

5. Increment the total seek count with this distance.

6. Currently serviced track position now becomes the new head position.

7. Go to step 3 until we reach one of the ends of the disk.

8. If we reach at the end of the disk reverse the direction and go to step 2 until all tracks in

request array has not been serviced.

**6. C-SCAN**

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 it 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.

1. 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.

2. The head services only in the right direction from 0 to size of the disk.

3. While moving in the left direction do not service any of the tracks.

4. When we reach at the beginning(left end) reverse the direction.

5. While moving in right direction it services all tracks one by one.

6. While moving in right direction calculate the absolute distance of the track from the head.

7. Increment the total seek count with this distance.

8. Currently serviced track position now becomes the new head position.

9. Go to step 6 until we reach at right end of the disk.

10. If we reach at the right end of the disk reverse the direction and go to step 3 until all tracks n request array have not been serviced.

**IMPLEMENTATION**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<stdbool.h>

#define HIGH 199

#define LOW 0

int choice,track,no\_req,head,head1,distance;

int disc\_req[100],finish[100];

void menu()

{

printf("\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*");

printf("\n\n1. SSTF \n2. SCAN \n3. C-LOOK \n4. FCFS \n5. C-SCAN \n6. LOOK \n7. Exit");

printf("\n\nEnter your choice: ");

scanf("%d",&choice);

}

void sstf()

{

char user;

int i, queue[100], queue2[100], q\_size, head, seek=0, temp;

float avg;

printf("\n\n-----SSTF Disk Scheduling Algorithm-----");

printf("\nEnter the size of the queue: ");

scanf("%d", &q\_size);

printf("Enter queue elements\n");

for( i = 0; i < q\_size; i++)

{

printf("Value No. [%d]: ", i + 1);

scanf("%d", &queue[i]);

}

printf( "Enter initial head position: ");

scanf("%d", &head);

//get distance from head of elems in queue

for(int i=0; i<q\_size; i++){

queue2[i] = abs(head-queue[i]);

}

//swap elems based on their distance from each other

for(int i=0; i<q\_size; i++){

for(int j=i+1; j<q\_size;j++){

if(queue2[i]>queue2[j]){

temp = queue2[i];

queue2[i]=queue[j];

queue2[j]=temp;

temp=queue[i];

queue[i]=queue[j];

queue[j]=temp;

}

}

}

for(int i=1; i<q\_size; i++){

seek = seek+abs(head-queue[i]);

head = queue[i];

}

printf("\nTotal seek time is %d\t",seek);

avg = seek/(float)q\_size;

printf("\nAverage seek time is %f\t", avg);

getch();

system("cls");

}

void scan()

{

int queue[20];

int head, max, q\_size, temp, sum;

int dloc; //location of disk (head) arr

printf("\n\n-----SCAN Disk Scheduling Algorithm-----\n");

printf( "Input no of disk locations: ");

scanf("%d", &q\_size);

printf( "Enter head position: ");

scanf("%d", &head);

printf( "Input elements into disk queue\n");

for(int i=0; i<q\_size; i++)

{

printf("Value No. [%d]: ", i + 1);

scanf("%d", &queue[i]);

}

queue[q\_size] = head; //add RW head into queue

q\_size++;

//sort the array

for(int i=0; i<q\_size;i++){

for(int j=i; j<q\_size; j++){

if(queue[i]>queue[j]){

temp = queue[i];

queue[i] = queue[j];

queue[j] = temp;

}

}

}

max = queue[q\_size-1];

//locate head in the queue

for(int i=0; i<q\_size; i++){

if(head == queue[i]){

dloc = i;

break;

}

}

if(abs(head-LOW) <= abs(head-HIGH)){

for(int j=dloc; j>=0; j--){

printf("%d --> ",queue[j]);

}

for(int j=dloc+1; j<q\_size; j++){

printf("%d --> ",queue[j]);

}

} else {

for(int j=dloc+1; j<q\_size; j++){

printf("%d --> ",queue[j]);

}

for(int j=dloc; j>=0; j--){

printf("%d --> ",queue[j]);

}

}

sum = head + max;

printf("\nmovement of total cylinders %d", sum);

getch();

system("cls");

}

void clook()

{

struct request;

{

int request\_track\_number;

bool visited;

}

int n, i, j, head, item[20], dst[20];

int cyl=0;

printf("\n\n-----CLOOK Disk Scheduling Algorithm-----\n");

printf("Enter no. of locations: ");

scanf("%d",&n);

printf("Enter position of head: ");

scanf("%d",&head);

printf("Enter elements of disk queue\n");

for(i=0; i<n; i++)

{

printf("Value No. [%d]: ", i + 1);

scanf("%d", &item[i]);

dst[i]=(head-item[i]);

}

//Selection Sort

for(i=0;i<n-1;i++)

{

for(j=i+1;j<n;j++)

{

if(dst[j]>dst[i])

{

int temp=dst[j];

dst[j]=dst[i];

dst[i]=temp;

temp=item[i];

item[i]=item[j];

item[j]=temp;

}

}

}

for(i=0;i<n;i++)

{

if(item[i]>=head)

{

j=i;

break;

}

}

printf("j=%d", j);

printf("\n\nOrder of disk allocation is as follows:\n");

for(i=j;i<n;i++)

{

printf(" -> %d", item[i]);

cyl+= abs(head-item[i]);

head=item[i];

}

for(i=0;i<j;i++)

{

printf(" -> %d", item[i]);

cyl+= abs(head-item[i]);

head=item[i];

}

printf("\n\nCylinder movement: %d\n\n", cyl );

getch();

system("cls");

}

void fcfs()

{

int queue[20],n,head,i,j,k,seek=0,max,diff;

float avg;

printf("\n\n-----FCFS Disk Scheduling Algorithm-----\n");

printf("Enter the max range of disk: ");

scanf("%d",&max);

printf("Enter the size of queue request: ");

scanf("%d",&n);

printf("Enter the queue of disk positions to be read\n");

for(i=0;i<=n;i++)

{

printf("Value No. [%d]: ", i + 1);

scanf("%d",&queue[i]);

}

printf("Enter the initial head position\n");

scanf("%d",&head);

queue[0]=head;

for(j=0;j<=n-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);

}

printf("Total seek time is %d\n",seek);

avg=seek/(float)n;

printf("Average seek time is %f\n",avg);

getch();

system("cls");

}

void cscan (){

int queue[20], q\_size, head, i,j, seek=0, diff, max, temp, queue1[20], queue2[20], temp1=0, temp2=0;

float avg;

printf("\n\n-----CSCAN Disk Scheduling Algorithm-----\n");

printf( "Input no of disk locations: ");

scanf("%d", &q\_size);

printf( "Enter initial head position: ");

scanf("%d", &head);

printf( "Enter disk positions to be read\n");

for(i=0; i<q\_size; i++) {

printf("Value No. [%d]: ", i + 1);

scanf("%d", &temp);

if(temp >= head){

queue1[temp1] = temp;

temp1++;

} else {

queue2[temp2] = temp;

temp2++;

}

}

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;

}

}

}

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);

}

printf("Total seek time is %d\n", seek);

avg = seek/(float)q\_size;

printf("Average seek time is %f\n", avg);

getch();

system("cls");

}

void look()

{

#define LOW 0

#define HIGH 199

int queue[20], head, q\_size, i,j, seek=0, diff, max, temp, queue1[20], queue2[20], temp1=0, temp2=0;

float avg;

printf("\n\n-----LOOK Disk Scheduling Algorithm-----\n");

printf( "Input the number of disk locations: ");

scanf("%d", &q\_size);

printf( "Enter initial head position: ");

scanf("%d", &head);

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

for(i=0; i<q\_size; i++){

printf("Value No. [%d]: ", i + 1);

scanf("%d", &temp);

if(temp >= head){

queue1[temp1] = temp;

temp1++;

} else {

queue2[temp2] = temp;

temp2++;

}

}

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;

}

}

}

if(abs(head-LOW) >= abs(head-HIGH)){

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);

getch();

system("cls");

}

int main()

{

system("color 07");

while(1)

{

menu();

switch(choice)

{

case 1: sstf();

break;

case 2: scan();

break;

case 3: clook();

break;

case 4: fcfs();

break;

case 5: cscan();

break;

case 6: look();

break;

case 7: exit(0);

break;

default:

printf("\n\nEnter valid choice");

break;

}

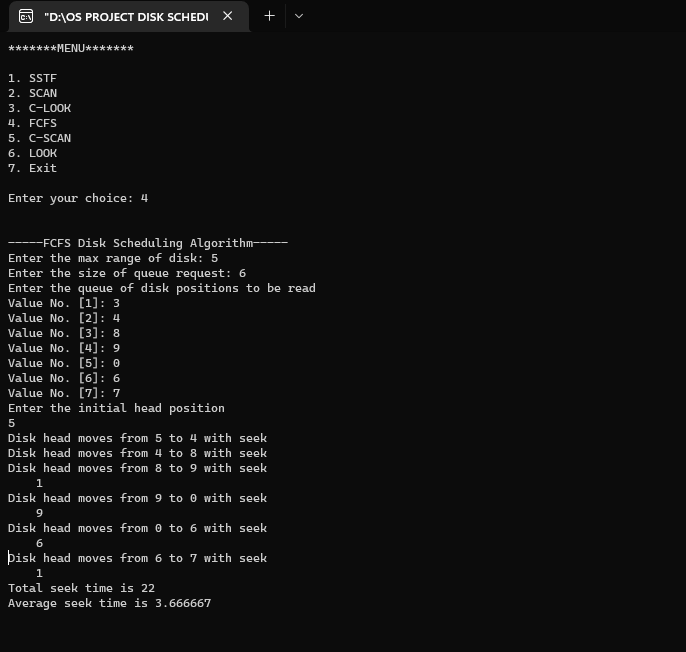
}

return 0;

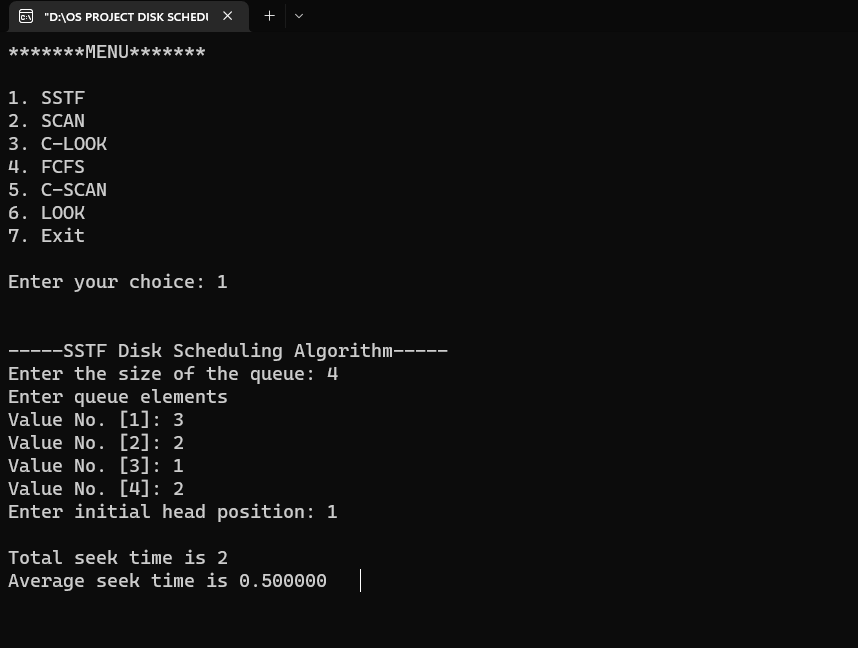
}

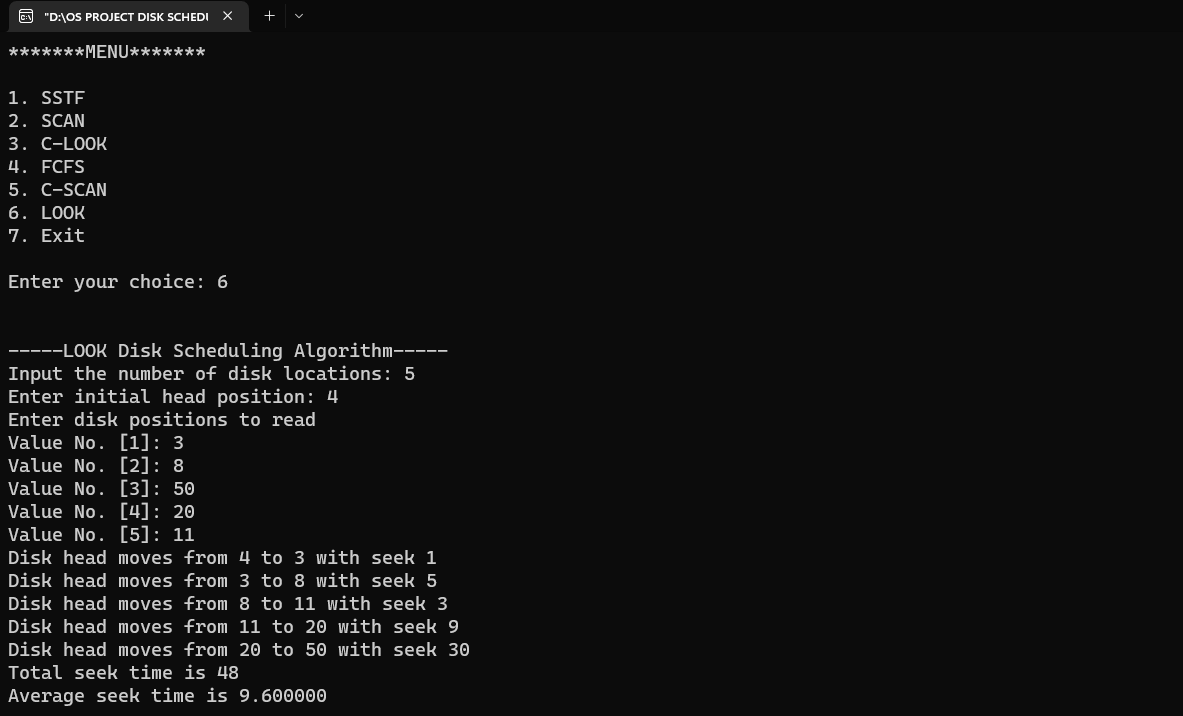
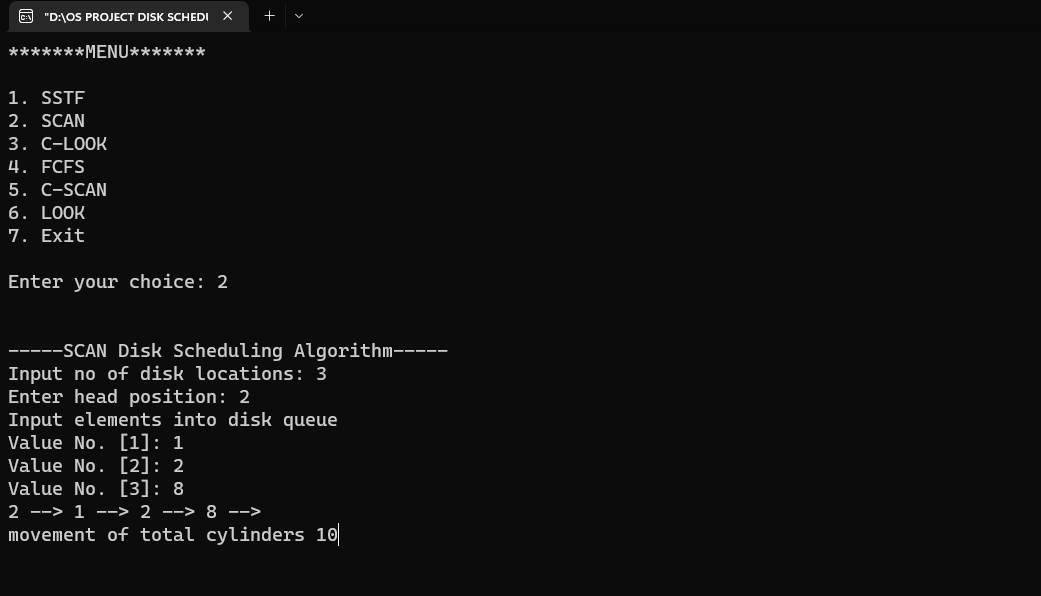
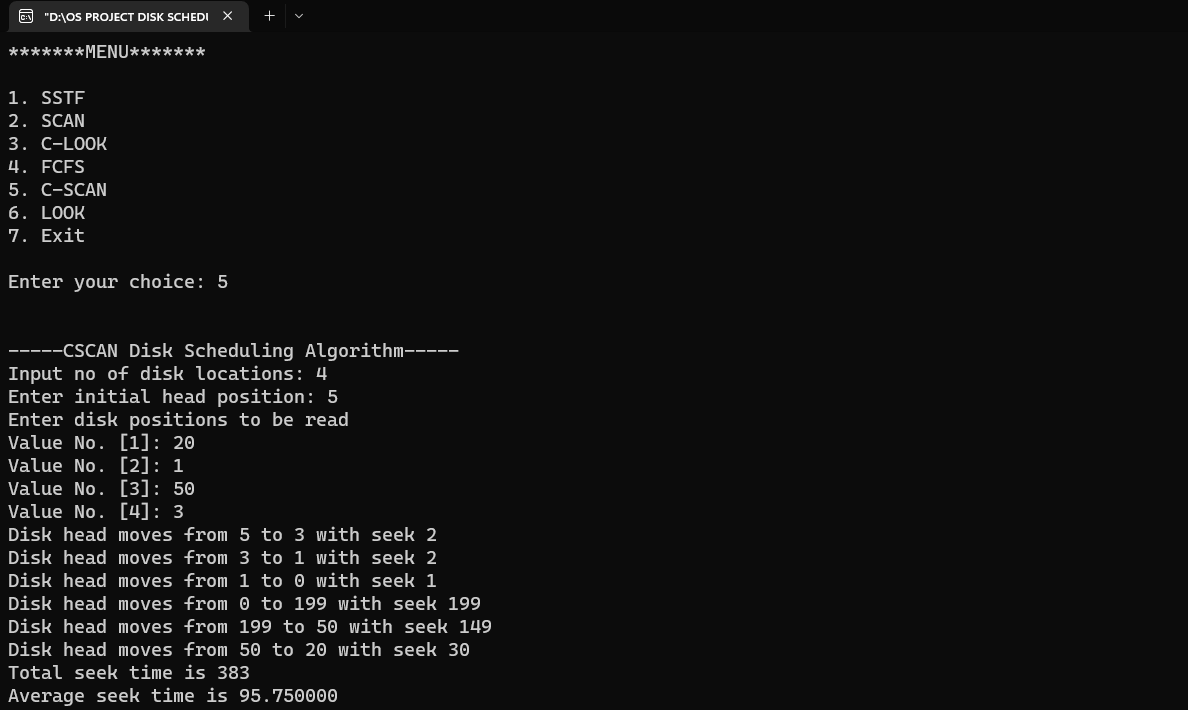
**OUTPUTS** **(TEST)**

1. **FCFS**

****

1. **SSTF**

****

1. **LOOK**
2. **C-LOOK**
3. **SCAN**
4. **C-SCAN**

**REFERENCES**

1. **JAVA POINT**

[**https://www.javatpoint.com/os-disk-scheduling#:~:text=The%20main%20purpose%20of%20disk,this%20request%20will%20be%20processed.**](https://www.javatpoint.com/os-disk-scheduling#:~:text=The%20main%20purpose%20of%20disk,this%20request%20will%20be%20processed.)

1. **GEEKS FOR GEEKS**

[**https://www.geeksforgeeks.org/disk-scheduling-algorithms/**](https://www.geeksforgeeks.org/disk-scheduling-algorithms/)

1. [**https://www.lnjpitchapra.in/wp-content/uploads/2020/04/file\_5e97ef5088ac0.pdf**](https://www.lnjpitchapra.in/wp-content/uploads/2020/04/file_5e97ef5088ac0.pdf)