

| **TITLE:** Implementation of Disk Scheduling Algorithm like FCFS, SSTF, SCAN, CSCAN, LOOK |
| --- |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** Implementation of Disk Scheduling Algorithm like FCFS, SSTF, SCAN, CSCAN, LOOK **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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. **Sumitabha Das “ UNIX Concepts & Applications”, McGraw Hill Second**

**Edition.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Knowledge of disk scheduling algorithm.

Calculation of seek time and transfer time etc \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Description of the application to be implemented**:

### 1. First Come-First Serve (FCFS)

* Definition: Processes requests in the order they arrive.
* Characteristics: Simple and fair but can lead to long wait times (convoy effect).
* Use Cases: Suitable for batch processing systems.

### 2. Shortest Seek Time First (SSTF)

* Definition: Serves the closest request to the current disk head position.
* Characteristics: Reduces total seek time but may cause starvation for distant requests.
* Use Cases: Effective in environments where quick access is essential.

### 3. Elevator (SCAN)

* Definition: Moves the disk head in one direction, servicing requests, then reverses direction.
* Characteristics: Balanced wait times and improved efficiency over FCFS.
* Use Cases: Ideal for general-purpose systems requiring fairness and efficiency.

### 4. C-SCAN (Circular SCAN)

* Definition: Similar to SCAN but jumps back to the start after reaching the end, servicing only in one direction.
* Characteristics: Provides uniform wait times and prevents starvation.
* Use Cases: Suitable for systems with large, uniform workloads.

### 5. LOOK

* Definition: Like SCAN, but only goes as far as the last request in each direction.
* Characteristics: Reduces unnecessary movement compared to SCAN.
* Use Cases: Efficient for systems needing responsive disk access.

**Implementation details:**

**FCFS:**

import matplotlib.pyplot as plt

def fcfs\_disk\_scheduling(requests, initial\_position):

current\_position = initial\_position

total\_movement = 0

order\_of\_requests = [initial\_position]

for request in requests:

movement = abs(request - current\_position)

total\_movement += movement

order\_of\_requests.append(request)

current\_position = request

return order\_of\_requests, total\_movement

def plot\_fcfs(order\_of\_requests):

plt.figure(figsize=(10, 6))

plt.plot(order\_of\_requests, marker='o', linestyle='-')

plt.title('Disk Scheduling - FCFS ')

plt.xlabel('Sequence')

plt.ylabel('Blocks')

plt.xticks(range(len(order\_of\_requests)), [f"B{i}" for i in range(len(order\_of\_requests))])

plt.grid()

plt.axhline(y=order\_of\_requests[0], color='r', linestyle='--', label='Initial Position')

plt.legend()

plt.show()

# Get user input

initial\_position = int(input("Enter the initial position of the head: "))

requests\_input = input("Enter the disk blocks separated by spaces: ")

requests = list(map(int, requests\_input.split()))

order, total\_movement = fcfs\_disk\_scheduling(requests, initial\_position)

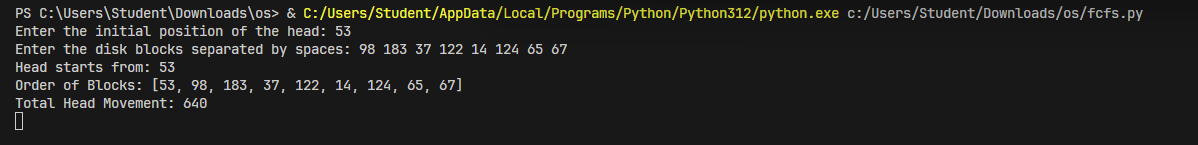
print("Head starts from:", initial\_position)

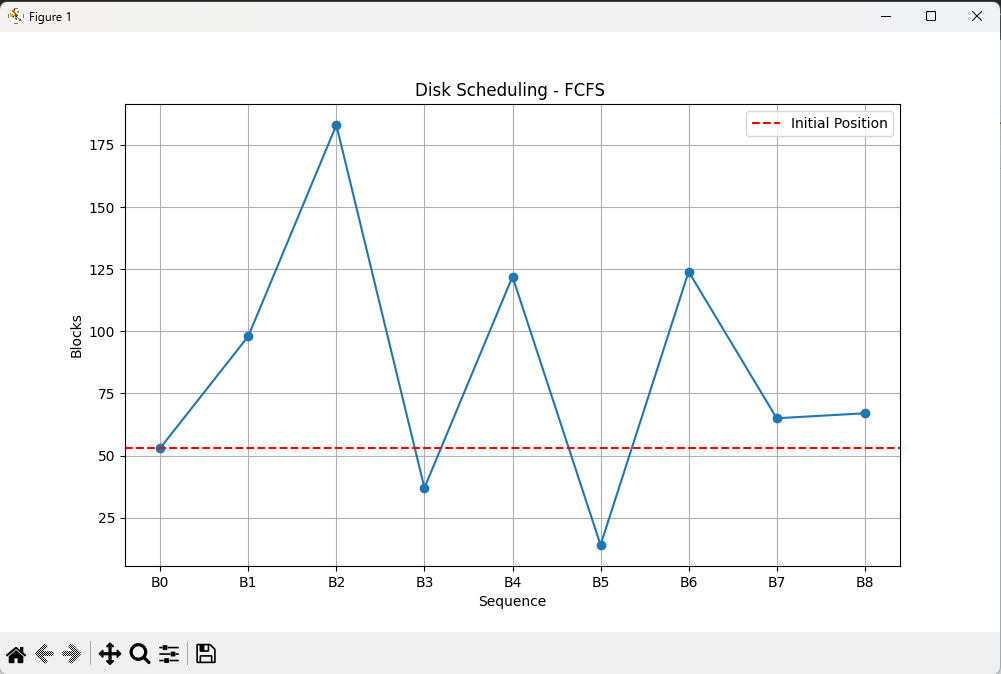
print("Order of Blocks:", order)

print("Total Head Movement:", total\_movement)

# Plot the graph

plot\_fcfs(order)





**C-SCAN :**

**import matplotlib.pyplot as plt**

**def c\_scan(requests, initial\_position, total\_cylinders, direction):**

**current\_position = initial\_position**

**total\_movement = 0**

**order\_of\_requests = []**

**# Sort the requests**

**requests.sort()**

**# Add the initial position to the order of requests**

**order\_of\_requests.append(current\_position)**

**if direction == 'up':**

**# Service requests in the upward direction**

**for request in requests:**

**if request >= current\_position:**

**movement = abs(request - current\_position)**

**total\_movement += movement**

**current\_position = request**

**order\_of\_requests.append(request)**

**# Jump to the beginning of the disk**

**if current\_position != total\_cylinders - 1:**

**total\_movement += abs(total\_cylinders - 1 - current\_position) # Move to end**

**current\_position = 0 # Jump to start**

**total\_movement += abs(total\_cylinders - 1) # Jump back to start**

**order\_of\_requests.append(0)**

**# Service remaining requests from start to the last processed**

**for request in requests:**

**if request < initial\_position:**

**movement = abs(request - current\_position)**

**total\_movement += movement**

**current\_position = request**

**order\_of\_requests.append(request)**

**elif direction == 'down':**

**# Service requests in the downward direction**

**for request in reversed(requests):**

**if request <= current\_position:**

**movement = abs(request - current\_position)**

**total\_movement += movement**

**current\_position = request**

**order\_of\_requests.append(request)**

**# Jump to the end of the disk**

**if current\_position != 0:**

**total\_movement += abs(current\_position) # Move to start**

**current\_position = total\_cylinders - 1 # Jump to end**

**total\_movement += abs(current\_position) # Jump back to end**

**order\_of\_requests.append(total\_cylinders - 1)**

**# Service remaining requests from end to the last processed**

**for request in reversed(requests):**

**if request > initial\_position:**

**movement = abs(request - current\_position)**

**total\_movement += movement**

**current\_position = request**

**order\_of\_requests.append(request)**

**return order\_of\_requests, total\_movement**

**def plot\_c\_scan(order\_of\_requests):**

**plt.figure(figsize=(10, 6))**

**plt.plot(order\_of\_requests, marker='o', linestyle='-', color='b')**

**plt.title('Disk Scheduling - C-SCAN')**

**plt.xlabel('Sequence')**

**plt.ylabel('Blocks')**

**plt.xticks(range(len(order\_of\_requests)), [f"B{i}" for i in range(len(order\_of\_requests))])**

**plt.grid()**

**plt.axhline(y=order\_of\_requests[0], color='r', linestyle='--', label='Initial Position')**

**plt.legend()**

**plt.show()**

**# Get user input**

**initial\_position = int(input("Enter the initial position of the head : "))**

**requests\_input = input("Enter the disk blocks separated by spaces: ")**

**requests = list(map(int, requests\_input.split()))**

**total\_cylinders = 200 # Fixed total cylinders**

**direction = input("Enter the direction of movement (199-up/0-down): ").strip().lower()**

**# Validate direction input**

**if direction not in ['up', 'down']:**

**print("Invalid direction. Please enter 'up' or 'down'.")**

**else:**

**order, total\_movement = c\_scan(requests, initial\_position, total\_cylinders, direction)**

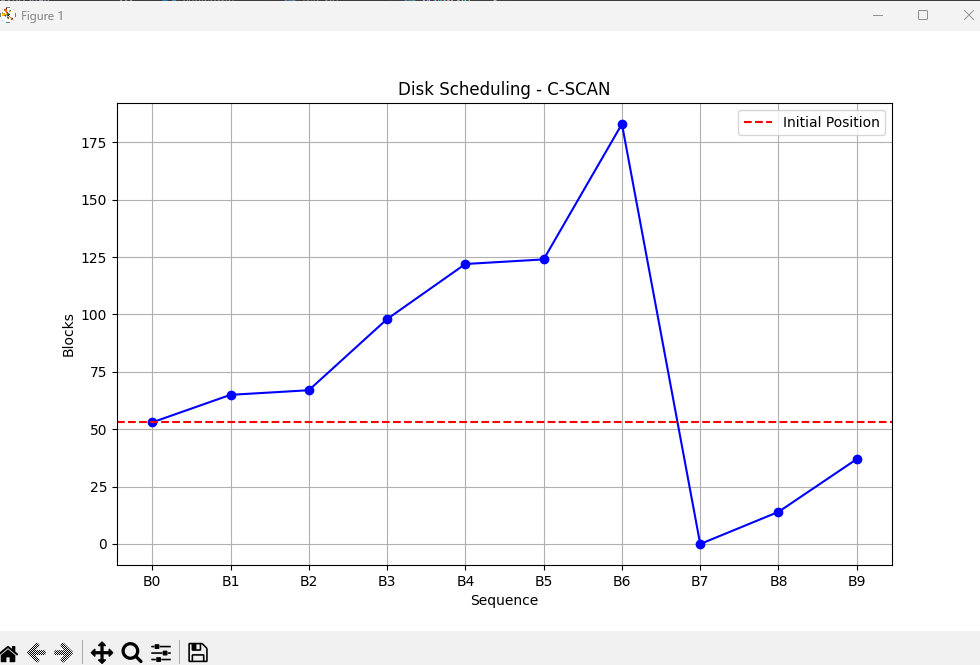
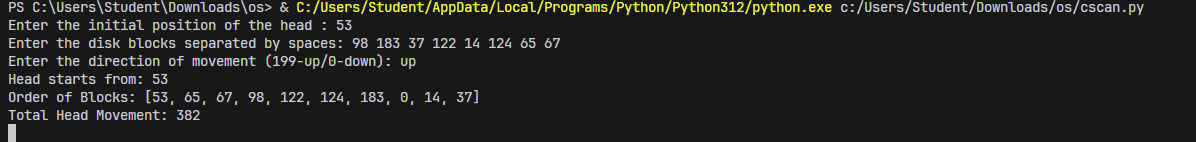
**print("Head starts from:", initial\_position)**

**print("Order of Blocks:", order)**

**print("Total Head Movement:", total\_movement)**

**# Plot the graph**

**plot\_c\_scan(order)**



**Conclusion:**

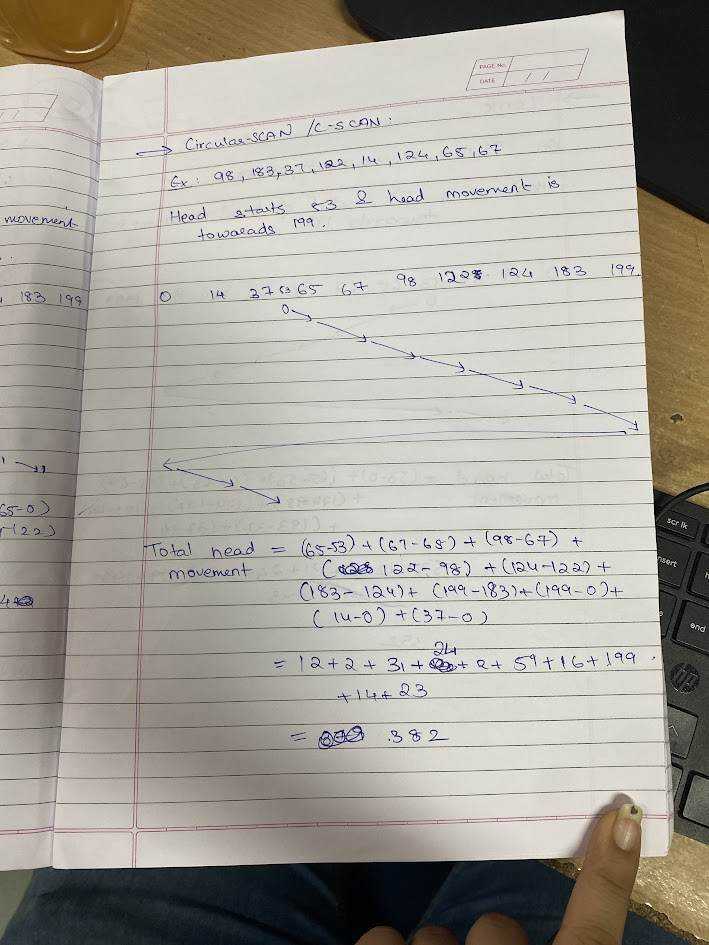
Learned Disk Scheduling Algorithms.

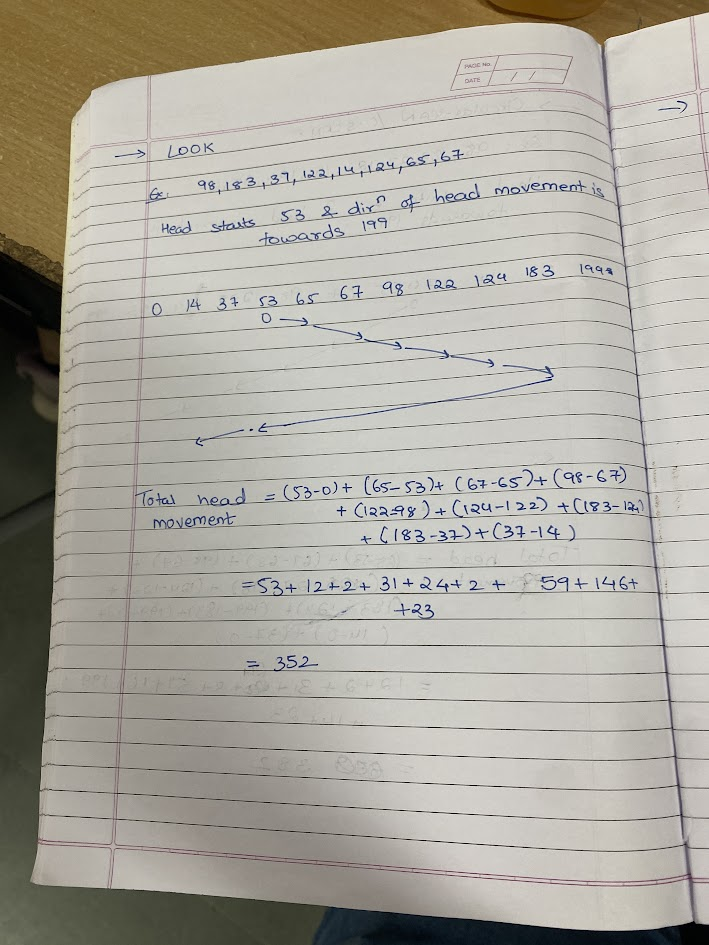
**Post Lab Descriptive Questions**

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.





**Post Lab Objective Questions**

1. In a hard disk, what rotates about a central spindle
   1. Disk
   2. Platter
   3. Sector
   4. None of the above

**Ans: Platter**

1. The time required to move the disk arm to the required track is known as
   1. Latency time
   2. Access time
   3. Seek time
   4. None of the above

**Ans: Seek time**

**Date: 24/10/24 Signature of faculty in-charge**