K. J. Somaiya School of Engineering, Mumbai-77
Somaiya Vidyavihar University
Department of Computer Engineering

Batch: A1	**Roll No.: 16010123012**
Experiment No. 09	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Implementation of Disk scheduling policies

AIM: Implementation of Disk scheduling algorithms - FCFS, SSTF, SCAN, CSCAN, LOOK, CLOOK (Any two) as instructed by instructor)

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:
The main purpose of 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

FCFS Scheduling Algorithm
It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

SSTF Scheduling Algorithm
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.

Scan Algorithm
It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path, and then it turns back and moves in the reverse direction satisfying requests coming in its path.

It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

Look Scheduling

It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.**:**

Assigned Algorithm 1 details: (FCFS)

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival.
2. Let the array size be n, arr[n]
3. ‘head’ is the initial position of disk head.
4. Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.
Cur_track=arr[i]
distance = abs(cur_track - head)
5. Initialize the seek count with Zero
6. Increment the total seek count with this distance.
7. Total seek=total seek+distance
8. Currently serviced track position now becomes the new head position.
Head=cur_track
9. Go to step 2 until all tracks in request array have not been serviced.

Assigned Algorithm 2 details: (CSCAN)

1. Let 'arr[n]' represent the array storing requested track numbers.
2. Let 'head' be the initial position of the disk head and let 'disk_size' be the total size of the disk (i.e., the max track number).
3. Add '0' and 'disk_size' to the request array to simulate circular movement.
4. Sort the updated request array in ascending order.
5. Find the index 'pos' of the first track in 'arr[]' such that 'arr[pos] >= head'.
6. Initialize 'total_seek = 0'.
7. Service all requests from 'pos' to end of the array:
   • For each 'i = pos' to 'n-1':
   - 'cur_track = arr[i]'
   - 'distance = abs(cur_track - head)'

```
   - 'total_seek = total_seek + distance'
   - 'head = cur_track'
8. After reaching the end, simulate jump to the start of the disk:
    - 'distance = abs(arr[0] - head)'
    - 'total_seek = total_seek + distance'
    - 'head = arr[0]'
9. Service remaining requests from start to 'pos - 1':
    • For each 'i = 1' to 'pos - 1':
    - 'cur_track = arr[i]'
    - 'distance = abs(cur_track - head)'
    - 'total_seek = total_seek + distance'
    - 'head = cur_track'
10. Output the sequence and 'total_seek'
```

Source code:

1. FCFS

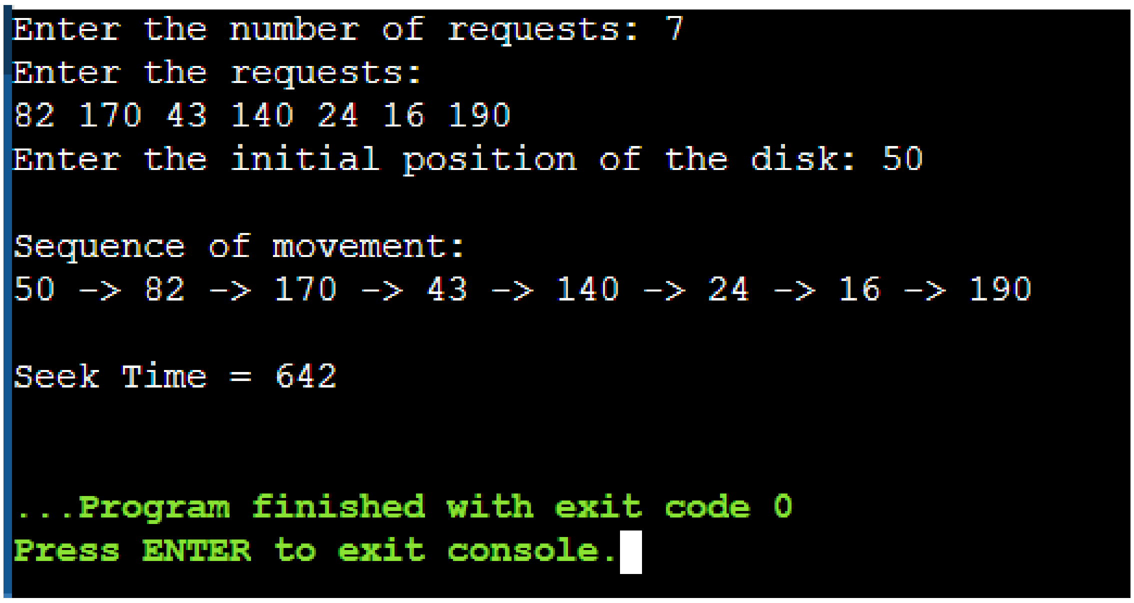
```cpp
#include <bits/stdc++.h>
#define endl '\n';
using namespace std;

int main()
{
  int n, head, move = 0;
  cout << "Enter the number of requests: ";
  cin >> n;
  vector<int> requests(n);
  cout << "Enter the requests: " << endl;
  for (int i = 0; i < n; ++i)
  {
    cin >> requests[i];
  }

  cout << "Enter the initial position of the disk: ";
  cin >> head;
  cout << "\nSequence of movement:" << endl;
  cout << head;
  for (int i = 0; i < n; ++i)
  {
    move += abs(requests[i] - head);
    head = requests[i];
    cout << " -> " << head;
  }
  cout << "\n\nSeek Time = " << move << endl;
  return 0;
}
```

2. CSCAN

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int main()
{
  int n, head, size, move = 0;

  cout << "Enter the number of requests: ";
  cin >> n;

  vector<int> requests(n);

  cout << "Enter the requests:" << endl;
  for (int i = 0; i < n; ++i)
  {
    cin >> requests[i];
  }

  cout << "Enter the initial position of the disk: ";
  cin >> head;

  cout << "Enter the total disk size: ";
  cin >> size;

  requests.push_back(0);
  requests.push_back(size);
  n += 2;

  sort(requests.begin(), requests.end());

  int pos = 0;
  while (pos < n && requests[pos] < head)
  {
    pos++;
  }

  cout << "\nSequence of head movement:" << endl;
  cout << head;

  for (int i = pos; i < n; ++i)
  {
    move += abs(head - requests[i]);
    head = requests[i];
    cout << " -> " << head;
  }

  move += abs(head - requests[0]);
  head = requests[0];
  cout << " -> " << head;

  for (int i = 1; i < pos; ++i)
  {
    move += abs(head - requests[i]);
    head = requests[i];
    cout << " -> " << head;
  }

  cout << "\n\nSeek Time = " << move;

  return 0;
}
```

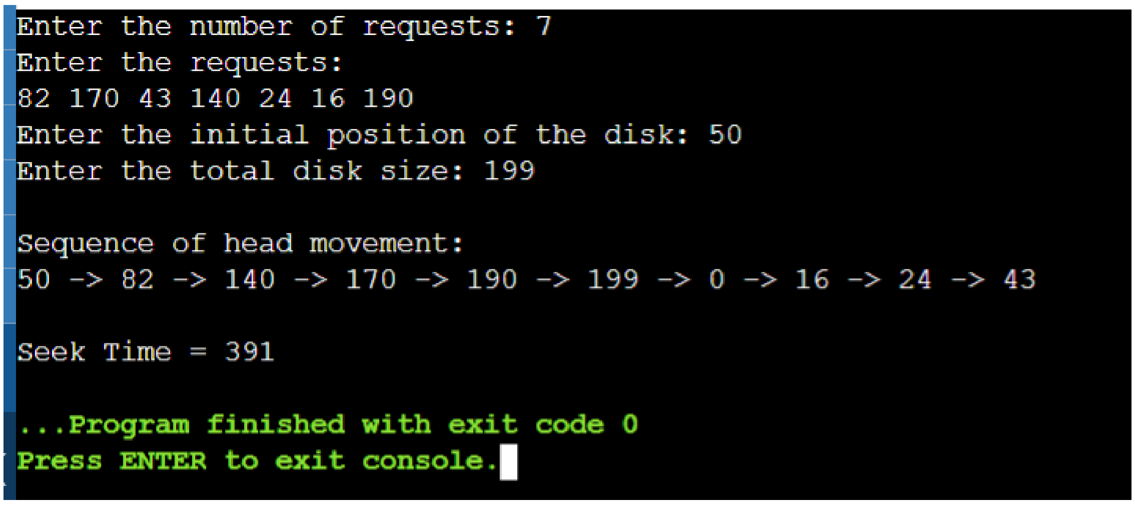
Output screenshots:

1. FCFS



```
Enter the number of requests: 7
Enter the requests:
82 170 43 140 24 16 190
Enter the initial position of the disk: 50

Sequence of movement:
50 -> 82 -> 170 -> 43 -> 140 -> 24 -> 16 -> 190

Seek Time = 642


...Program finished with exit code 0
Press ENTER to exit console.
```

2. CSCAN



```
Enter the number of requests: 7
Enter the requests:
82 170 43 140 24 16 190
Enter the initial position of the disk: 50
Enter the total disk size: 199

Sequence of head movement:
50 -> 82 -> 140 -> 170 -> 190 -> 199 -> 0 -> 16 -> 24 -> 43

Seek Time = 391

...Program finished with exit code 0
Press ENTER to exit console.
```

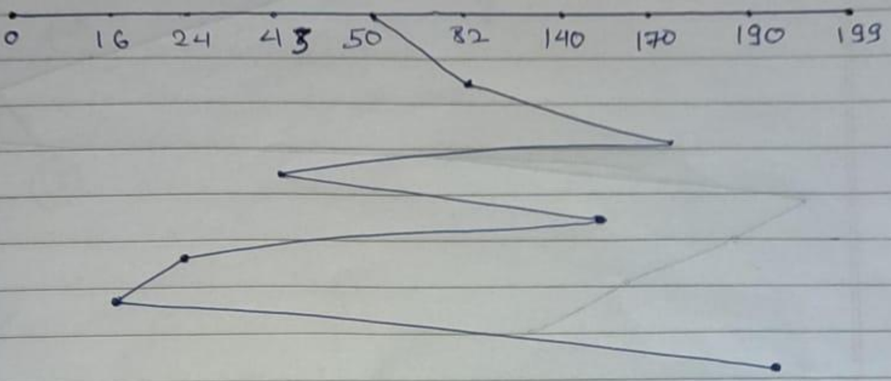
Conclusion:

I have successfully completed the implementation and analysis of disk scheduling algorithms—FCFS and CSCAN. Through this experiment, I understood how different scheduling policies affect the total seek time and the efficiency of servicing disk I/O requests. FCFS was simple and easy to implement but not always optimal in seek time, whereas CSCAN provided a more uniform wait time by servicing requests in one direction and jumping to the start, effectively minimizing the variance in response time. This practical understanding deepened my knowledge of operating system concepts related to disk management and performance optimization.

Lab Work:

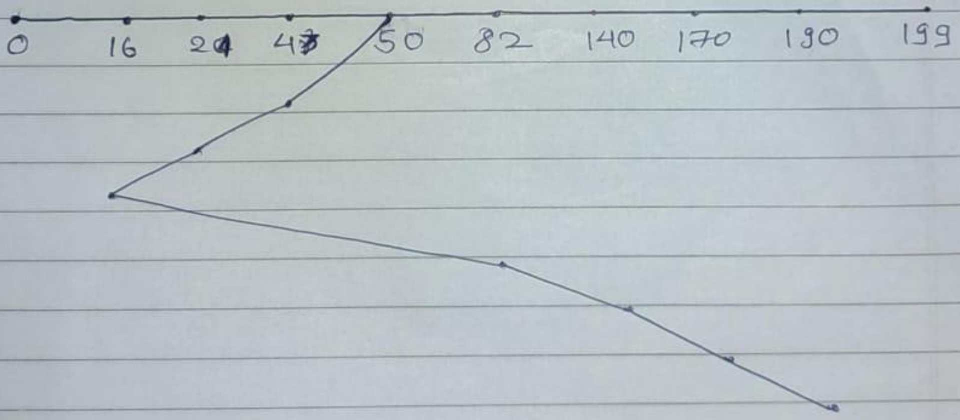
• Disk Sheduling Algorithms

1) FCFS (First come First serve)
Order of Request - 82, 170, 43, 140, 24, 16, 190
Current position - 50

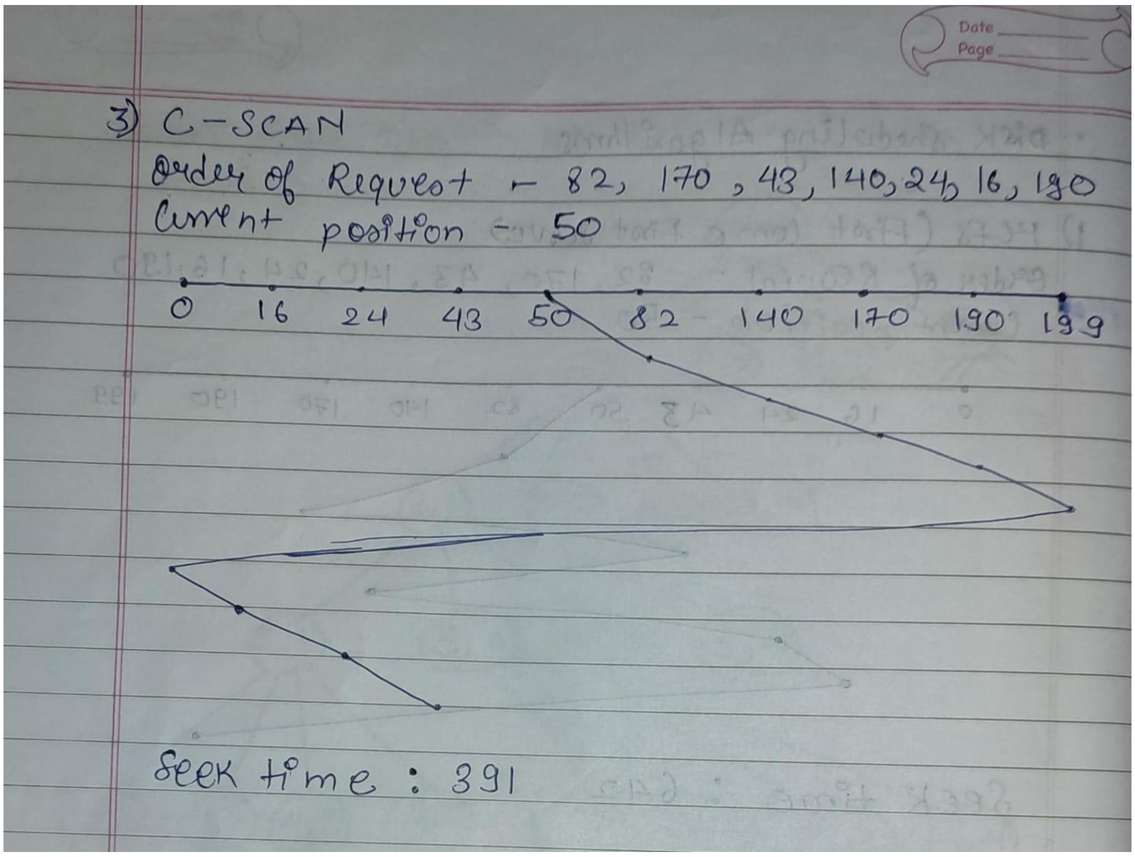


Seek time : 642

2) SSTF (Shortest seek Time First)
Order of Request - 82, 170, 43, 140, 24, 16, 190
Current position - 50



seek time : 208



Post Lab Descriptive Questions

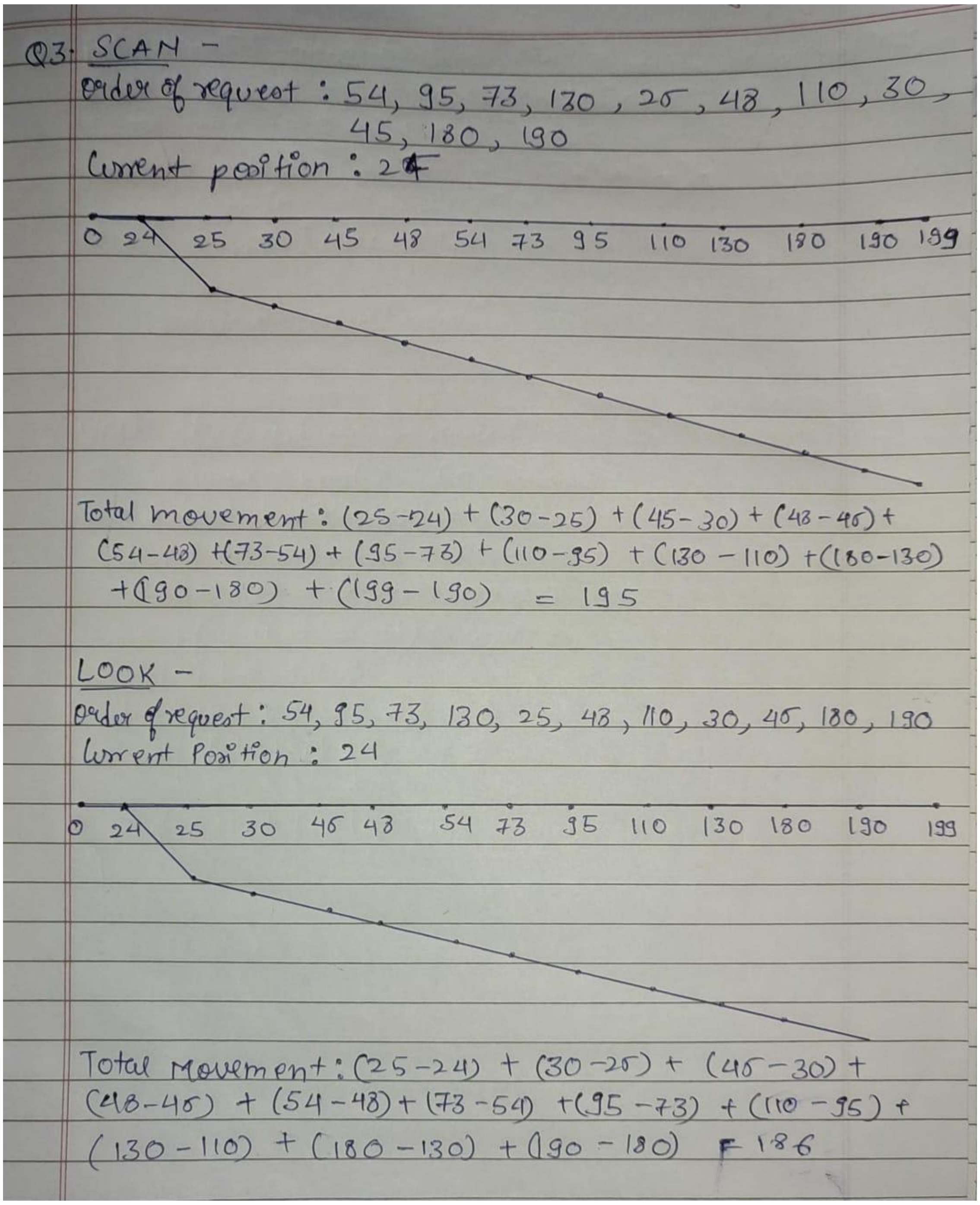
i. Explain how SSTF can lead to starvation. Can this problem occur in SCAN or LOOK? Justify your answer.

The SSTF (Shortest Seek Time First) disk scheduling algorithm can lead to starvation because it always selects the request closest to the current head position. While this approach minimizes seek time, it can cause requests that are farther away to be postponed indefinitely if new, nearer requests keep arriving. As a result, some disk I/O requests may never get serviced, leading to starvation. On the other hand, SCAN and LOOK algorithms prevent this problem. In SCAN, the disk head moves in one direction servicing requests until it reaches the end, then reverses, ensuring all requests are eventually handled. LOOK functions similarly but only goes as far as the last request in each direction before turning back. Since both SCAN and LOOK ensure that the head covers all positions in a regular pattern, no request waits indefinitely, and hence, starvation does not occur.

ii. **How does the size of the disk (number of tracks) influence the performance of the SCAN, C-SCAN, and LOOK algorithms? Provide examples to support your answer.**

The size of the disk, particularly the number of tracks, directly impacts the performance of SCAN, C-SCAN, and LOOK algorithms by affecting the average seek time and efficiency. As the number of tracks increases, the disk arm needs to travel longer distances, potentially increasing the delay in servicing requests. In SCAN, the head moves from one end of the disk to the other and back, so a larger disk results in longer travel and wait times; for example, moving from track 10 to 190 on a 200-track disk takes more time than on a smaller disk. C-SCAN, which services requests in one direction and jumps back to the beginning without servicing any in reverse, also suffers performance drops on larger disks, as the return sweep becomes longer—e.g., a request at track 20 may wait longer on a 500-track disk. In contrast, LOOK and C-LOOK improve efficiency on larger disks by moving only as far as the last request in a given direction, avoiding unnecessary traversal across unused tracks. This makes LOOK-based algorithms more scalable and efficient for disks with a high number of tracks.

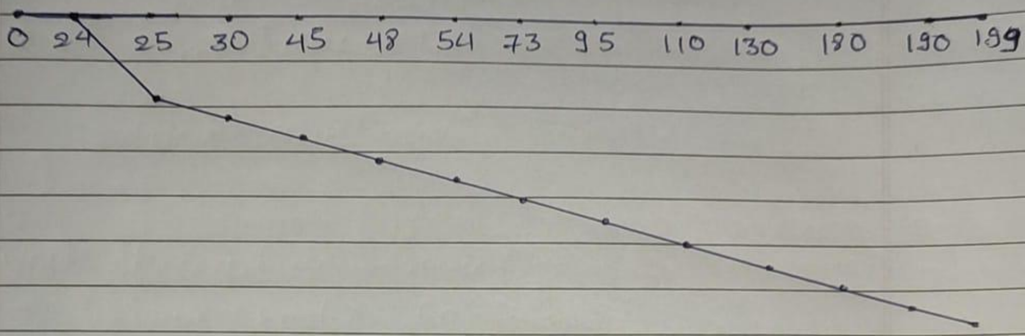
iii. **Suppose that a disk drive has 200 cylinders numbered 0 to 199. Consider a disk queue with I/O requests on the following cylinders in their arriving order: 54, 95, 73, 130, 25, 48, 110, 30, 45, 180, and 190. The disk head is assumed to be at cylinder 24. Starting from the current head position, what is the total distance (in cylinders) the disk arm moves to satisfy all the pending requests for the SCAN and LOOK disk-scheduling algorithm?**



Q3. SCAN -

Order of request : 54, 95, 73, 130, 25, 48, 110, 30, 45, 180, 190

Current position : 24

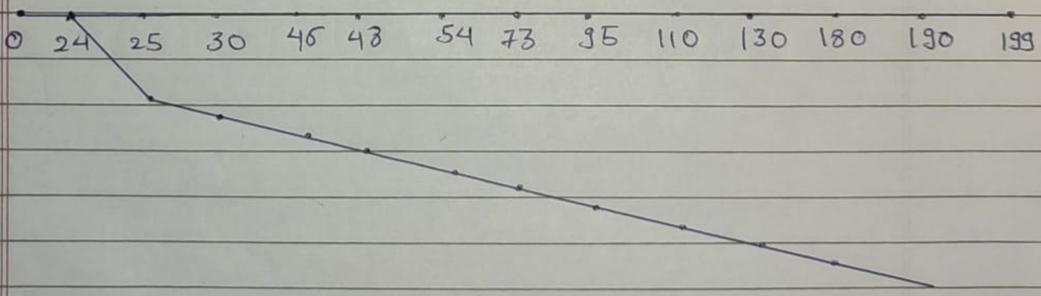


Total movement : (25-24) + (30-25) + (45-30) + (48-45) + (54-48) + (73-54) + (95-73) + (110-95) + (130-110) + (180-130) + (190-180) + (199-190) = 195

LOOK -

Order of request : 54, 95, 73, 130, 25, 48, 110, 30, 45, 180, 190

Current Position : 24



Total Movement : (25-24) + (30-25) + (45-30) + (48-45) + (54-48) + (73-54) + (95-73) + (110-95) + (130-110) + (180-130) + (190-180) = 186

Date: 08 / 04 / 2025