# CHAPTER-1

INTRODUCTION

## 1.1. Background:

The management of disk performance is an important aspect of an operating system. Since the speed of processor and main memory have been increased several times than the speed of the disk, the difference in the speed of processor and the disk, I/O performance of disk has become an important bottleneck. In any disk system with a moving read/write head, the seek time between cylinders takes a significant amount of time. This seek time should be minimized to better access time.

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/O request is going to be fulfilled earliest is called disk scheduling algorithm. The main objectives for any disk scheduling algorithm are minimizing the response time and maximizing the throughput.

### 1.1.1 Disk Scheduling Criteria:

Generally, a set of criteria is established against which various scheduling policies are evaluated.

### 1.1.2. Seek Time:

The time for the disk arm to move the heads to the cylinder containing the desired sector.

### 1.1.3. Rotational latency:

The additional time incurred for the disk to rotate the desired sector to the disk head.

### 1.1.4. Disk bandwidth:

Total number of bytes transferred divided by the total time between the first request for service and the completion of last transfer.

### 1.1.5 Transfer Time:

The time required for the transfer, this depends on the rotational speed of the disk.

## 1.2. Disk Scheduling Algorithms:

There are several disk-scheduling algorithms such as FCFS, SSTF, SCAN, C-SCAN, LOOK and C-LOOK algorithms, which helps in scheduling the I/O request.

### 1.2.1. First Come First Serve (FCFS) Scheduling:

This is the simplest algorithm among others. In this scheduling algorithm, all incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served i.e. the I/O request that arrived first is served first. Other requests are served as per their time of arrival.

### 1.2.2. Shortest Seek Time First (SSTF) Scheduling:

In this case, the request is serviced according to the next shortest distance. The next requests are selected which have the minimum seek time from the current head position.

### 1.2.3. SCAN Scheduling:

This approach works like an elevator. Disk arm starts at one end of the disk and scans towards the nearest end, servicing the entire request it finds out when moves towards the other end and then when it hits the other end, it scans reversely servicing the requests that it didn't get earlier. If a request comes in after it has been scanned, it will not be serviced until the head arm comes back

.

### 1.2.4. C-SCAN Scheduling:

Circular SCAN (C-SCAN) is a variant of SCAN algorithm. It begins its scan towards the nearest end and works its way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. The huge jump doesn't count as a head movement.

### 1.2.5. LOOK Scheduling:

This approach is equivalent to SCAN algorithm, except that the arm goes only as far as the last request in each direction without going all the way to end. It can be considered as directional algorithm as it reverses the direction as soon as it served the last request in a particular direction.

### 1.2.6. C-LOOK Scheduling:

This is just an enhanced version of C-SCAN. In this, the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end.

## 1.3. Objectives:

Following are the objectives of the work:

- Study and analyze different disk scheduling algorithms.
- To calculate total disk head movement and average seek time of different queue in different scheduling algorithms.
- To prepare simulator and implement the different algorithms using simulator.
- To compare the average seek time of different algorithms and recommend the better algorithm.

# CHAPTER-2
# Literature Survey

## 2.1. Literature Survey:

One of the main goal of the operating system for the disk drives is to use the hardware efficiently, we can meet this goal using fast access time and large disk bandwidth that depends on the relative positions of the read-write head and the requested data. Since memory management allows multiprogramming so that operating system keeps several read/write request in the memory. In order to service these requests, hardware (disk drive and controller) must be used efficiently. To support this in disk drive, the hardware must be available to service the request. If the hardware is busy, we can't service the request immediately and the new request will be placed in the queue of pending requests. Several disk scheduling algorithms are available to service the pending requests. Among these disk scheduling algorithms, the algorithm that yields less number of head movement will remain has an efficient algorithm .

Since almost all the computer resources are needed to be scheduled before use, scheduling is one of the important fundamental function of operating system. Disk is one of the important resources of computer. For meeting the responsibilities for the disk driver entailshaving large disk bandwidth and fast access time. Speed of the processor and the memory capacity are increasing at very fast rate over 40% per year but the speed of disk is growing only 7% per year.

Disk scheduling algorithms are used to allocate the services to the I/O requests on the disk. Since seeking disk requests is time consuming, disk scheduling algorithms try to minimize this latency. If desired disk drive or controller is available, request is served immediately. When one request is completed, the Operating System has to choose which pending request to service next. The OS relies on the type of algorithm it needs when dealing and choosing what particular disk request is to be processed next. The objective of using these algorithms is keeping head movements to the amount as possible. The less the head to move, the faster the seek time will be.

First Come First Serve is considered as the simplest one algorithm of disk scheduling. It focuses the execution of processes in a sequential manner. In other words, we can say that in this type of scheduling, process is coming in an order and they are getting the attention in that particular order only that is a sequential order. FCFS has the advantage of being simple but it is unable to produce efficient results in all the situation.

SSTF as its name itself describes its working principle. This algorithm considers the shortest seek time instead of the fact that the process is not in sequential order. Requests that are comparatively in near position with reference to the current disk head location as compared to the requests that are at distant location, will have the service first.

Elevator algorithm is another popular name of the SCAN algorithm because its working principle is much similar to an elevator. In this scheme, the disk head starts moving from one end. When all the requests of this direction gets processed, the disk head changes its direction and stats travelling to the other end and provide its services to all requests on its way.

LOOK algorithm's working principle is influenced with Scan algorithm. The head traverse across the disk surface in both directions to perform read & write operation. In Look algorithm, disk head stats its trip from the current head position to the one end of the disk servicing all the requests on its way. As it reaches the last request in that particular direction, changes its direction immediately without going to the extreme end unlike SCAN algorithm, which touches the innermost and outermost cylinders in each direction. C-SCAN and C-LOOK algorithm is also there.

The researcher has studied the different research papers related to the articles Comparative Analysis of Disk Scheduling Algorithm. By examining the different factors which affect the seek time of disk head movement in different scheduling algorithms the researcher has drawn a conclusion and suggested the better algorithm for disk scheduling.

### 2.1.1. Analysis of Previous Research Works:

While doing the research, the previous research works should be analyzed. It is a process used by researchers for narrow down the research to the specific point which helps in reducing a large chunk of data into smaller fragments, which makes sense. The previous research works analysis also gives the direction for the further research. After the analysis of previous research, data, process, the research can find the research gap and start new research from that gap.

Many research has been done in the topic "Disk Scheduling Algorithms" and many new algorithms are formed on the basis of previous research. Most of the researchers have done the comparative analysis of the different disk scheduling algorithms and gave conclusions. The performance analysis of various disk scheduling algorithms have been done based on various factors with the I/O request 100, 180, 40, 120, 20, 130,60, 70. They assumed a 100000 RPM disk has 8 heads and 480 cylinders which is divided into 120 – cylinder zones with the cylinders in different zones containing 200, 240 ,280 and 320 sectors. They also assumed the seek time 4ms. From the comparative analysis, it is determined that Smallest Seek Time First (SSTF) disk scheduling algorithm produce better throughput among from remaining disk scheduling algorithm[8].

The detailed process for calculation of total head movement of FCFS, SSTF, SCAN and C-SCAN algorithms have been explained by giving the I/O request 50, 91, 150, 42, 130, 18, 140, 70, 60 and assumed that the disk head is initially at cylinder 50 and came in conclusion that most widely used approaches to improve disk input-output performance is disk scheduling[9].

The comparative study of disk scheduling algorithms have been done and analyzed different disk scheduling algorithms: FCFS, SSTF, SCAN, LOOK ,C- SCAN, C-LOOK, FSCAN, N-Step SCAN with the I/O request (0-199): 60, 143, 15, 185, 85, 120, 33, 28, 146 having read/write head initially at cylinder 70 and gave the conclusion that different algorithms have their own different features and as per situation they are implemented to maximize the performance of the system[1]. Another analysis have been performed on different disk scheduling algorithms with the I/O request (0-199): 95, 180, 34, 119, 11, 123, 62, 64. In case of C- SCAN and C-LOOK, this paper assumed the value α the distance travelled from one end point to another end point (i.e. in circular we go from initial head position to final head position directly) as 20ms[6]. Where, in other papers the value α is not mentioned and the actual distance have been taken to calculate the disk head movement.

A discussion have been performed on "A comprehensive review for Disk Scheduling Algorithms"and tries to show the distinctive qualities of the different scheduling algorithms with the I/O request (0-199): 95, 180, 34, 119, 11, 123, 62, 64, initial R/W head is at 50. They have given the distinctive features of different algorithms. The FCFS performs operations in order requested. No reordering of work queue since it processed disk requests according to its arrival. There is no starvation and all the requests are serviced but it doesn't provide fastest service. The Shortest Seek Time First (SSTF) selects the disk I/O request that requires the least movement of the disk access arm from its current position regardless of direction. It also reduces the seek time compared to FCFS but in this algorithm, I/O requests at the edges of the disk surface may get starved. The SCAN algorithm go from the outside to the inside servicing requests and then back from the outside to the inside servicing requests. It also reduces variance compared to SSTF. The Circular Scan (C-SCAN) moves from one end of the disk

to the other, servicing requests. When other end is reached, it immediately returns to the beginning of the disk, without servicing any requests. This algorithm treats the cylinders as a circular list that wraps around from the last cylinder to the first one. It also provides a more uniform wait time than SCAN. In LOOK scheduling algorithm, the arm goes only as far as the final request in each direction. The direction reverses immediately, without going all the way to the end of the disk. The Circular LOOK (C- LOOK) algorithm is similar to C-SCAN. The disk head also goes as far as the last request in its direction then reverses its direction immediately without first going all the way to the end of the disk[10].

A proposal on Hybridized Disk Scheduling Algorithm (HDSA) with two disk I/O requests having queue (10-199): 23, 89, 132, 42, 187 with current head at 100 and another queue (10-199): 36, 80, 120, 10, 15, 40, 188, 150, 168 with current head position at 130, compared the total head movement of the new algorithm with the existing algorithm and came on conclusion that: Hybridized Disk Scheduling Algorithm (HDSA) shows better performance than existing conventional disk scheduling algorithms such as FCFS, SSTF, SCAN, C- SCAN, LOOK and C-LOOK. The result of the experiment showed that the number of head movement of the HDSA was reduced compared to the conventional disk scheduling algorithms. This increase efficiency of the disk performance in computer resource management[11].

The Disk Scheduling Algorithm Simulator have been developed which is written in java and designed as a means of enhancing students' learning of disk scheduling algorithms. The simulator has three modes: Simulation Mode where, a set of cylinder numbers to be accessed in to the disk requests text box are specified. Practice Mode, where, the user can decide which request will be serviced next by clicking on the cylinder number the user thinks should be accessed next and the user click the answer button to check if the answer is right. Comparison mode, where, the scheduling algorithms can be evaluated by running them on a particular set of disk requests and computing the total seek length, the total seek time and the average seek time. The user can compare the performance of all the algorithms at the same time. This simulator presents an intuitive, engaging and easy-to-use simulator that animates the concepts of traditional disk scheduling algorithms. The students can work, experiment and do more scheduling problems with the simulator[12].

Another algorithm is proposed as "A zone based improved disk scheduling algorithms" with different I/O requests and compare total head movement of different algorithms with proposed algorithm. They came to the conclusion that: The proposed algorithm provides performance fairness to perform various requests of disk and shows that it has better average access time than former methods[7].

"A New Heuristic Disk Scheduling Algorithm" with the queue (0-199): 36, 80, 120, 10, 15, 40, 188, 150 and 186. (When starting head position is 130, 20 and 155) have been proposed. They have compared the average head movement of different algorithm with proposed algorithm and came in the conclusion that: From the experiment and comparison of proposed algorithm with existing algorithm it is clear that proposed algorithm reduces head movement and decide best route[13].

"Disk Scheduling Algorithm" have been proposed by assigning the priorities for different cylinder number as: 98 (6), 183 (8), 37 (1), 122 (5), 14 (2), 124 (7), 65 (3), 67 (4), compared the total head movement of existing algorithm with proposed algorithm with some assumptions: Each Read/Write request must be assign with some priority, the disk request which has assigned highest priority that can be serviced first, the disk head is not moving till end of the disk and is moved further towards the request and the disk requests can be processed in both directions i.e. either in forward direction and backward direction. After the comparison they gave the conclusion as: It is proved very easily that this scheduling algorithm is better than some of the existing algorithms[14].

"A New Optimized Real-Time Disk Scheduling Algorithm" have been proposed with the request queue (0-200): 38, 180, 130, 10, 50, 15, 190, 90 and 150 with initial head at 120. They have done the comparative analysis of various classical existing disks scheduling algorithms and newly developed optimized real time disk scheduling algorithm has been performed on the basis of number of head movement and they came to conclusion that: With the help of experiments and comparison of proposed algorithm with existing algorithms, it is clear that the proposed algorithm reduces the total head movement[5].

"Major Half Served First (MHSF) Disk Scheduling Algorithm" have been proposed with the three cases of I/O requests as: Case-I – 15, 50, 35, 22, 5, 12 with current head position at 30. Case II – 15, 10, 90, 75, 100, 80, 65, 5 with current head position at 55. Case III – 20, 30, 5, 95, 85, 55, 90, 100, 25, 15 with current head position at 45. He has compared three algorithms MHSF, SSTF and FCFS and came on conclusion that: Experimental result shows that the proposed MHSF disk scheduling algorithm is giving better performance than SSTF and FCFS disk scheduling algorithms. The average seek time has been reduced which increases the efficiency of the disk performance[15]. "Optimized Disk Scheduling Algorithm (ODSA)" have been designed and evaluated the performance with the I/O requests of three cases: Case – I: 25, 10, 151, 170, 62, 46, 74 and 111. Initial head is at 45, minimum track 0 and maximum track 180. Case – II: 16, 75, 24, 21, 30, 80, 116 and 63 with initial head at 66.

Case – III: 25, 33, 54, 64, 40, 90, 110 and 160 with initial head is at 125. They have compared the average seek time and transfer time of proposed algorithm with existing algorithms and came on conclusion as: The proposed ODSA algorithm shows better performance than other disk scheduling algorithms (FIFO, SSTF, SCAN, C-SCAN and LOOK). The average seek time and transfer time has been improvised by this algorithm which increases the efficiency of the disk performance[16].

"An Improved Approach to Maximize the Performance of Disk Scheduling Algorithm by Minimizing the Head Movement and Seek Time using Sort Mid Current Comparison (SMCC)" have been proposed with the I/O requests of three cases: Case-I: I/O request (0-199): 98, 183, 37, 122, 14, 124, 65, 67 with initial head 53. Case – II: I/O request (0-99): 33, 72, 47, 8, 99, 74, 52, 75 with initial head 63. Case – III:I/O request (0-4999): 86, 1470, 913, 1774, 948, 1509,1022, 1750, 130 with initial head at 143. They have done the comparison of average seek time and total head movement of proposed algorithm with FCFS, SSTF, SCAN, C-SCAN, LOOL, C-LOOK and came in conclusion that: Compared to the classical approach of disk scheduling algorithm, the result and calculations shows that proposed algorithm reduces the number of head movement and seek time thus improving the performance of disk bandwidth for disk drives[4]. "An Improved FCFS (IFCFS) Disk Scheduling Algorithm" have been proposed and described an improvement of FCFS with I/O requests of three cases: Case-I: 80, 50, 30, 40, 5, 10 with initial head at 20. Case – II: 100, 5, 50, 90, 75, 10, 80, 60 with initial head at 40. Case-III: 20, 35, 5, 95, 75, 55, 85, 45, 40, 15 with initial head at 50. He compared the total head movement and average seek time of FCFS and proposed algorithm and came in conclusion that: The result shows that the proposed IFCFS disk scheduling algorithm is always giving better performance than FCFS[17].

# CHAPTER-3
# SYSTEM ANALYSIS

## 3.1. EXISTING SYSTEM:

It has been a part of our education system for as long we canremember In the existing system, we can store all the record manually that require large manpower &place to store all the records this system was carried out through a manual process.

### 3.1.1. DRAWBACKS IN EXISTING SYSTEM:

- Maintenance of records is difficult.
- Chance of occurrence of errors.
- Involves large amount of paperwork.
- Slowupdating & renewal of data.

## 3.2. PROPOSED SYSTEM:

Automated To overcome the disadvantages of the existing system we proposed the GUI system. The best technology learning tool such as interactive software are used by us to provide consuming the time

### 3.2.1. Advantages:

- Students are encouraged to think critically & support their option
- The automated system is time-saving and better performance than manual system
- No fever of data loss.
- Just need a little knowledge to operate the system.

# CHAPTER-4
# SYSTEM REQUIREMENTS

## 4.1. SOFTWARE REQUIREMENTS:

| | |
|---|---|
| Operating System | Windows 10 |
| Programming Language | Java |
| Frame Work | Eclipse IDE |

## 4.2 HARDWARE REQUIREMENTS:

| | |
|---|---|
| Processor | Pentium IV |
| Clock Speed | 2.86GHZ Processor |
| Hard disk | 4GB |
| RAM | 4GB RAM |

# CHAPTER-5
# Present work of the project

## 5.1. Present work of the project:

Disk scheduling algorithms 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. The procedures of these algorithms are described here:

### 5.1.1. First Come First Serve Disk Scheduling Algorithm (FCFS):

The functioning of this algorithm is maintained by First in First out (FIFO) queue. With this scheme, the I/O requests are served or processed according to their arrival [4]. Though this algorithm improves response time but fails to decrease the average seek time because this algorithm needs a lot of random head movements and disk rotations. Let us consider the following track requests in the disk queue to compute the total head movement of the read/write head and let us assume that the current read/write head position is at location 100 for a 200 track disk (0-199).

**Track Requests:**

25, 90, 135, 50, 190 and 60.

In First Come First Serve algorithm, the read/write head initially move from current location 100 to 25, then 25 to 90, then 90 to 135, then 135 to 50, then 50 to 190 and at last 190 to 60.

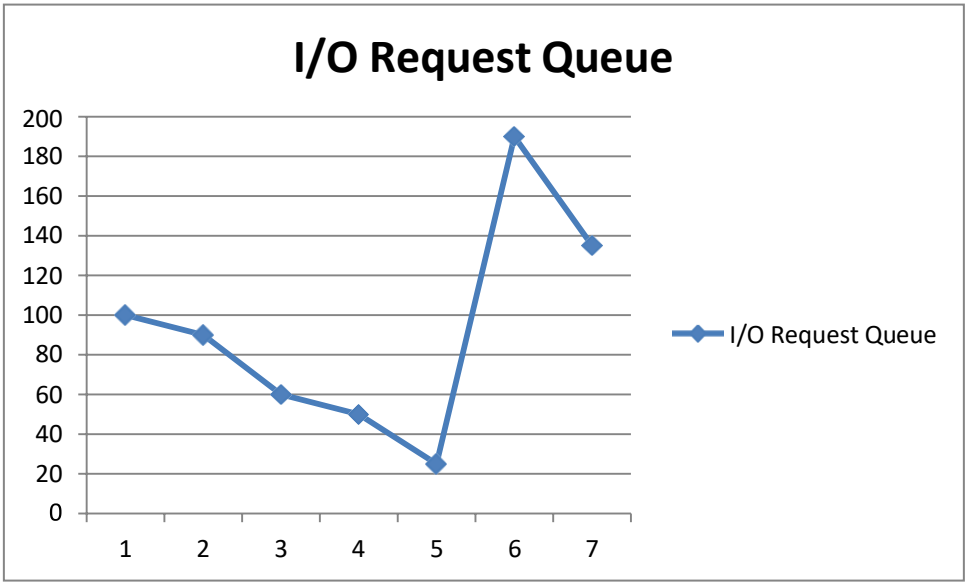


Figure5.1.1: First Come First Serve Representation

Total Head Movement = (100-25) + (90-25) + (135-90) + (135-50) + (190-50) + (190-60)= 540 tracks. The major disadvantage of FCFS disk scheduling algorithm is that it raises the mean seek time because disk arm has to cover long distance to accomplish a request as in this case movement of arm from 135 to 50 and then from 50 to 190.

### 5.1.2. Shortest Seek Time First Algorithm (SSTF):

In this approach, the read/write head serves the request first that have minimum seek time from the current head position. The I/O requests that are close to the read/write head are serviced first. SSTF disk scheduling algorithm is actually a form of SJF scheduling algorithm and may cause starvation of some requests [5]. 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. Let us consider the same track requests in the disk queue to compute the total head movement of the read/write head and let us again assume that the current read/write head position is at location 100 for a 200 track disk (0-199).

**Track Requests:**

25, 90, 135, 50, 190 and 60.

In Shortest Seek Time First algorithm, the read/write head initially move from current location 100 to 90, then 90 to 60, then 60 to 50, then 50 to 25, then 25 to 135 and at last 135 to 190.

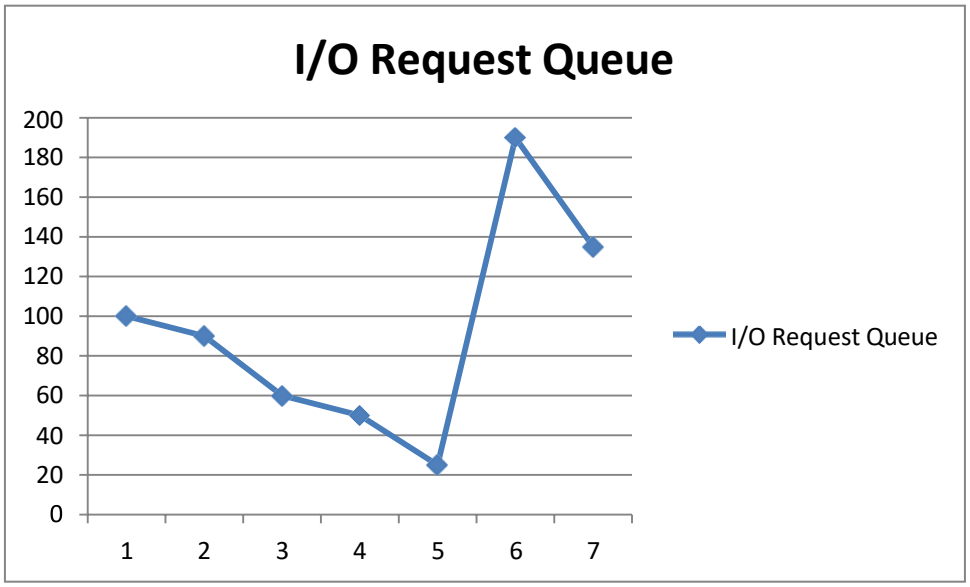


Figure 5.1.2: Shortest Seek Time First Representation.

Total Head Movement = (100-90) + (90-60) + (60-50) + (50-25) + (135-25) + (190-135) = 240 tracks. The major disadvantage of SSTF disk scheduling algorithm is that some requests have to wait for a long time if new requests with shorter seek time keep arriving but the advantage is that the throughput is higher as compared to FIFO disk scheduling algorithm and produces less head movements.

### 5.1.3. Scan Disk Scheduling Algorithm (SCAN):

In Scan disk scheduling algorithm, the read/write head starts from one end and move towards the other end and servicing requests as it reaches each track until it reaches to other 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 [6]. Let us consider the same track requests in the disk queue to compute the total head movement of the read/write head and let us again assume that the current read/write head position is at location 100 for a 200 track disk (0-199).

**Track Requests:**

25, 90, 135, 50, 190 and 60.

In Scan disk scheduling algorithm, the read/write head initially move from current location 100 to 90, then 90 to 60, then 60 to 50, then 50 to 25, then 25 to 0 and then to the other direction from 0 to 135 and at last 135 to 190. It services all the requests at either side firstly then moves the other way that's why it is sometimes called as elevator algorithm.

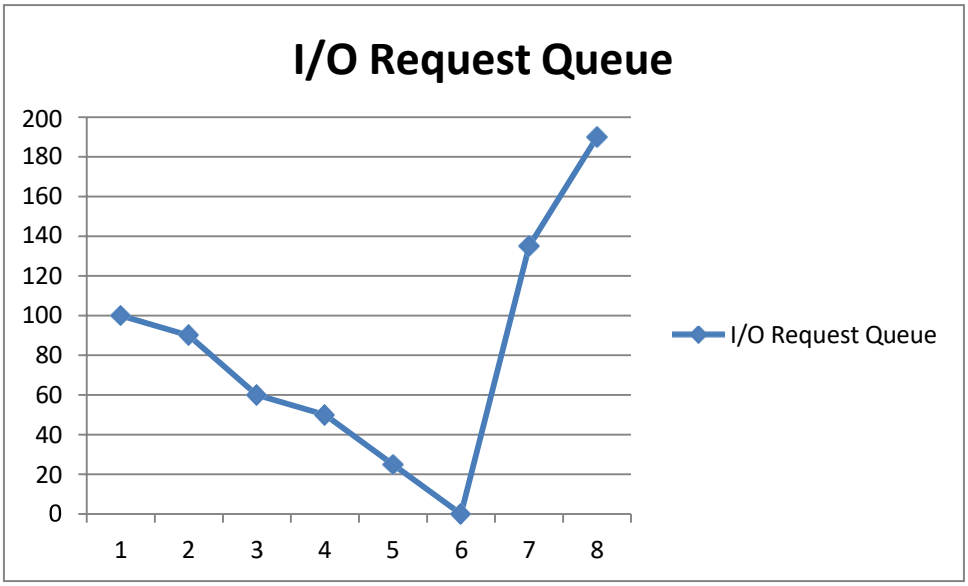


Figure5.1.3: SCAN Representation.

Total Head Movement = (100-90) + (90-60) + (60-50) + (50-25) + (25-0) + (135-0) +(190-135) = 290 tracks. The major disadvantage of Scan disk scheduling algorithm is that disk arm always starts from the beginning towards one end even other numbers of requests are present on the other end. The throughput of Scan algorithm is better than FCFS and also prevents starvation.

## 5.1.4. Look Disk Scheduling Algorithm (LOOK):

Look disk scheduling algorithm is the improved version of Scan disk scheduling algorithm and avoids the starvation problem. In Look disk scheduling algorithm, the arm goes only as 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. Let us consider the same track requests in the disk queue to compute the total head movement of the read/write head and let us again assume that the current read/write head position is at location 100 for a 200 track disk (0-199).

**Track Requests:**

25, 90, 135, 50, 190 and 60.

In Look disk scheduling algorithm, the read/write head initially move from current location 100 to 90, then 90 to 60, then 60 to 50, then 50 to 25, then to the other direction from 25 to 135 and then at last from 135 to 190.

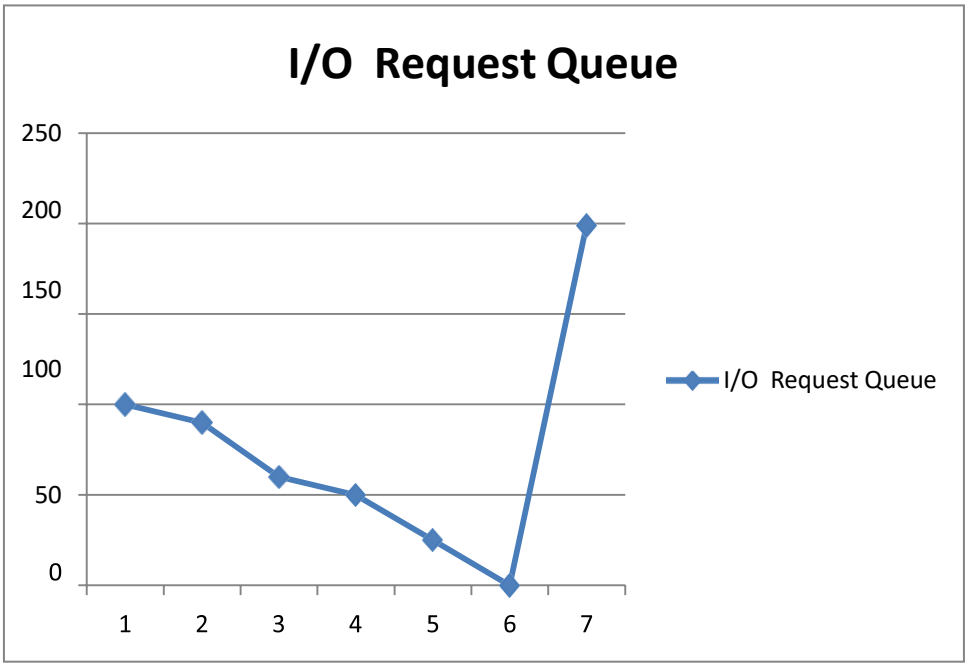


Figure 5.1.4: LOOK Representation.

Total Head Movement = (100-90) + (90-60) + (60-50) + (50-25) + (135-25) + (190-135)= 240 tracks. The major advantage of Look disk scheduling algorithm is that it improves response time and throughput than Scan disk scheduling algorithm.

## 5.1.5. 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 [8] Let us consider the same track requests in the disk queue to compute the total head movement of the read/write head and let us again assume that the current read/write head position is at location 100 for a 200 track disk (0- 199).

**Track Requests:**

25, 90, 135, 50, 190 and 60.

In C-Scan disk scheduling algorithm, the read/write head initially move from current location 100 to 90, then 90 to 60, then 60 to 50, then 50 to 25, then 25 to 0 and then to the other direction from 0 to 199, then 199 to 190 and at last 190 to 135. It services all the requests at either side firstly up to the end and then moves the other end's without fulfilling any requests in the way and then start servicing as shown in the figure below.

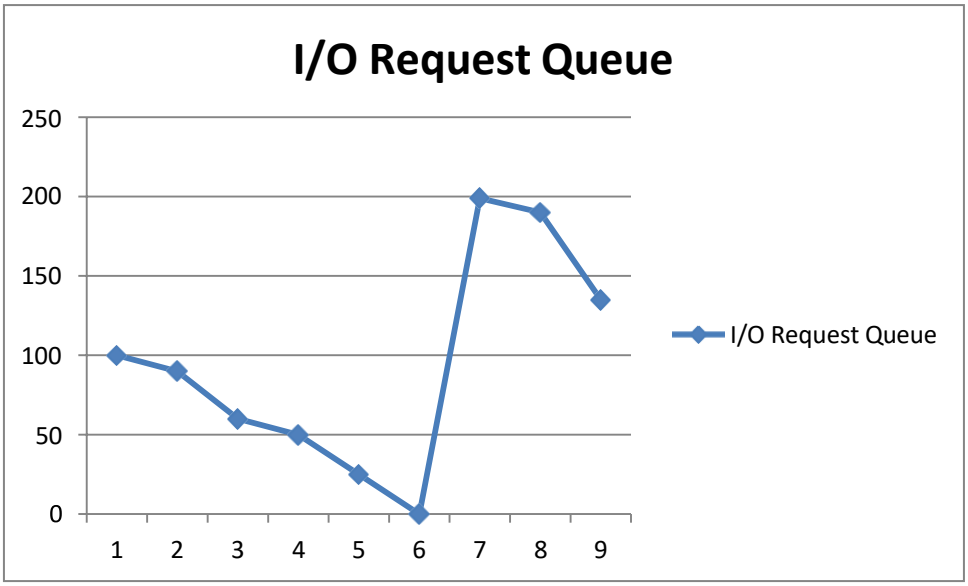


Figure 5.1.5: C-Scan Representation.

Total Head Movement = (100-90) + (90-60) + (60-50) + (50-25) + (25-0) + (199-0) +(199-190) + (190-135) = 363 tracks. The C-Scan disk scheduling algorithm satisfies requests only when the head moves in one direction and not satisfying any requests when it moves back.

## 5.1.6. Circular Look Disk Scheduling Algorithm (C-LOOK):

Cyclic Look or Circular Look disk scheduling algorithm is an improved version of C-Scan disk scheduling algorithm. Arm goes only as far as the last request in each direction and then reverses direction immediately without first going all the way to the end of the disk . Let us consider the same

track requests in the disk queue to compute the total head movement of the read/write head and let us again assume that the current read/write head position is at location 100 for a 200 track disk (0-199).

**Track Requests:**

25, 90, 135, 50, 190 and 60.

In C-Look disk scheduling algorithm, the read/write head initially move from current location 100 to 90, then 90 to 60, then 60 to 50, then 50 to 25 and then to the other direction from 25 to 190 and at last 190 to 135. It services all the requests at either side firstly up to the last request and then moves the other end's last request without fulfilling any requests in the way and then start servicing as shown in the figure below.

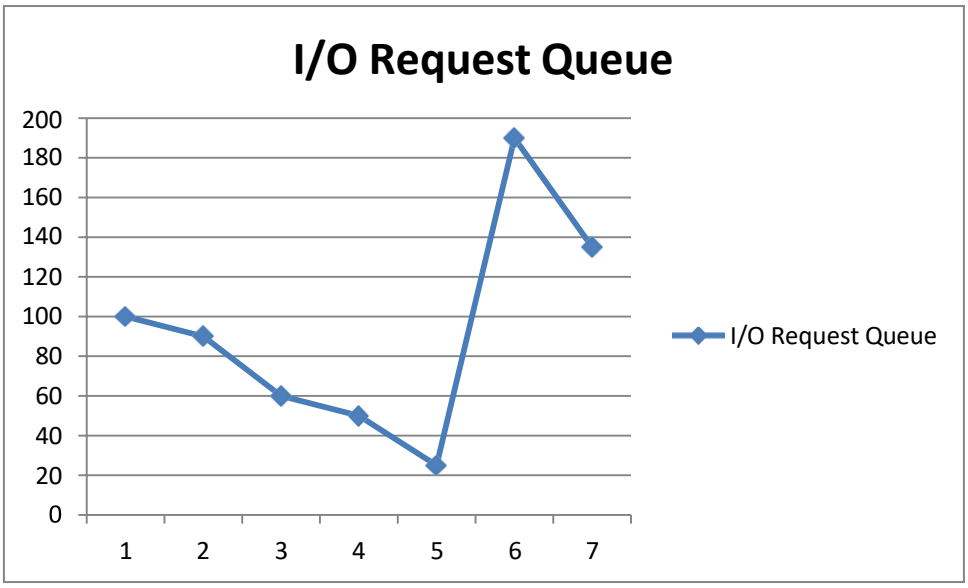


Figure 5.1.6: C-LOOK Representation

Total Head Movement = (100-90) + (90-60) + (60-50) + (50-25) + (190-25) + (190-135) = 295 tracks. The major advantage of C-Look disk scheduling algorithm is that it results in higher throughput and lower response time

## 5.2. PERFORMANCE OF DISK SCHEDULING IN C LANGUAGE:

In the stage of implementation of the research project, the researcher gives the clear picture of the research task and the process through which the researcher sought to reach the goal of the research. The researcher has used experimental research methodology in this research. The researcher has completed the literature review and prepared the sample I/O request queue on the basis of the literature review. At the initial phase, 5 requests are selected and manually calculated the Total head movement and Average seek time.

The researcher has formulated the five sample I/O requests having different size with different cases and have calculated total head movement and average seek time of different algorithms (FCFS, SSTF, SCAN, C-SCAN, LOOK and C-LOOK) and finally have calculated the average seek time of all algorithms.

### 5.2.1. Using FCFS:

Suppose the head of a moving – head disk with 200 tracks numbered 0 to 199) is currently serving request at tracks 130. The queue it requests is kept in the FIFO order: 36, 80, 120, 10, 15, 40, 188, 150, 168.Calculating the average seek time using different algorithms.

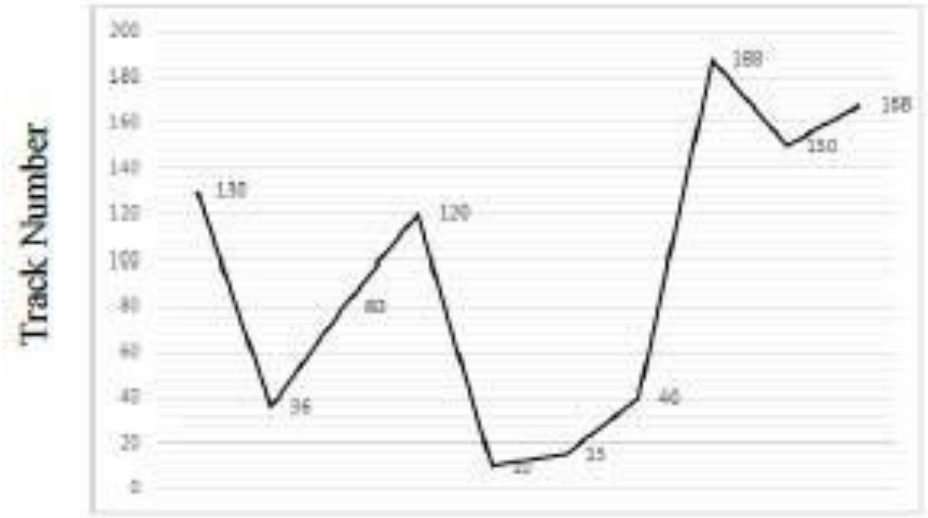


Figure5.2.1: FCFS [c]

Total head movement = |130-36| + |36-80| + |80-120| + |120-10| + |10-15| + |15-40| + |40-188| + |188-150| +|150-168| = 522

Average Seek Time = 522 / 9 ms = 58 ms.

### 5.2.2. Using SSTF:

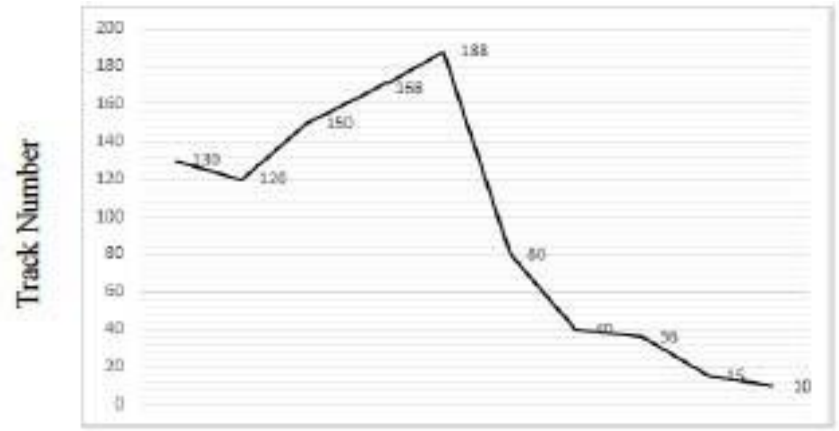


Figure5.2.2: SSTF[c]

Total head movement = |130-120| + |120-150| + |150- 168| + |168-188| + |188-80| + |80-40| + |40-36| + |36-15| + |15-10| = 256

Average Seek Time = 256 / 9 ms = 28.44 ms.

### 5.2.3. Using SCAN:

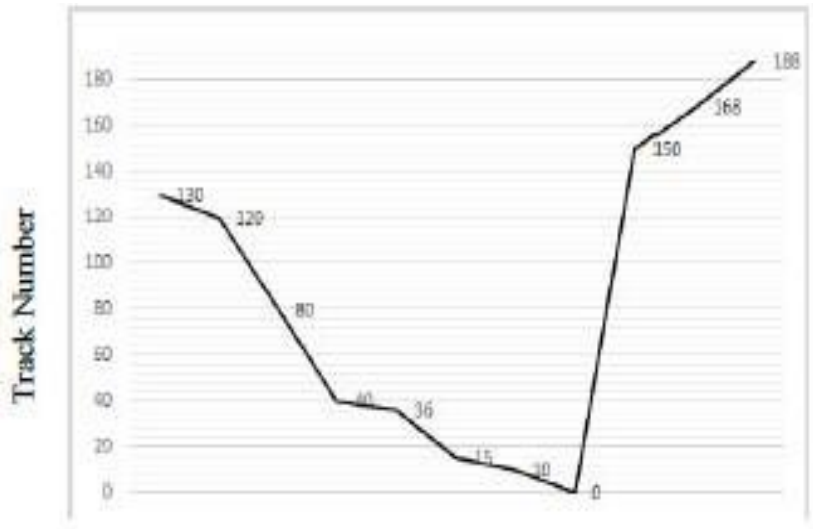


Figure5.2.3: SCAN[c]

Total head movement = |130-120| + |120-80| + |80-40| + |40-36| + |36-15| + |15-10| + |10-0| + |0-150| + |150-168| + |168-188| = 318

Average Seek Time = 318 / 9 ms = 35.33 ms.

### 5.2.4. Using C-SCAN:

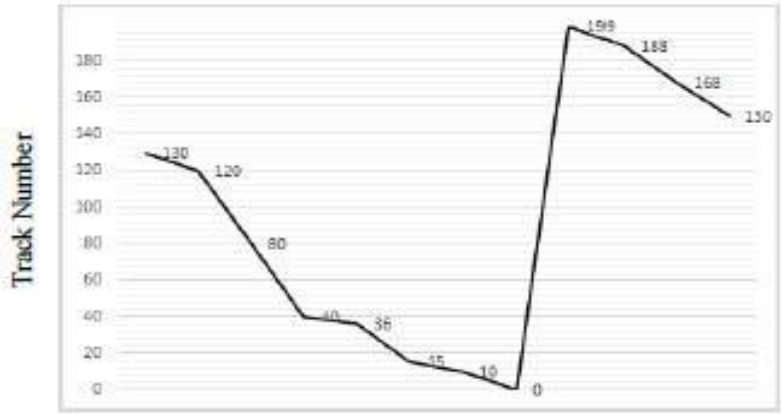


Figure5.2.4: C-SCAN[c]

Total head movement = |130-120| + |120-80| + |80-40| + |40-36| + |36-15| + |15-10| + |10-0| + |199-188| + |188- 168| + |168-150| = 179

Average Seek Time = 179 / 9 ms = 19.88 ms.

### 5.2.5. Using LOOK:

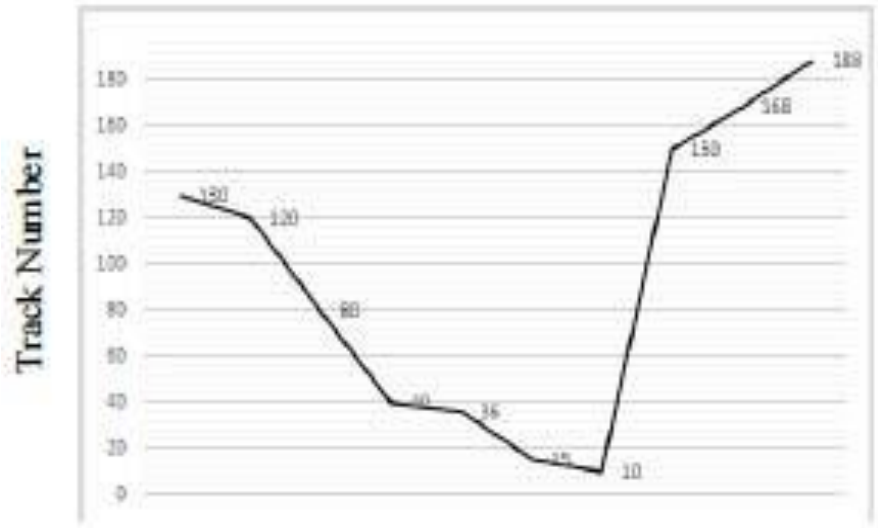


Figure5.2.5: LOOK[c]

Total head movement = |130-120| + |120-80| + |80-40| + |40-36| + |36-15| + |15-10| + |10-150| +|150-168| + |168- 188| = 298

Average Seek Time = 298 / 9 ms = 33.11 ms.

### 5.2.6. Using C-LOOK:

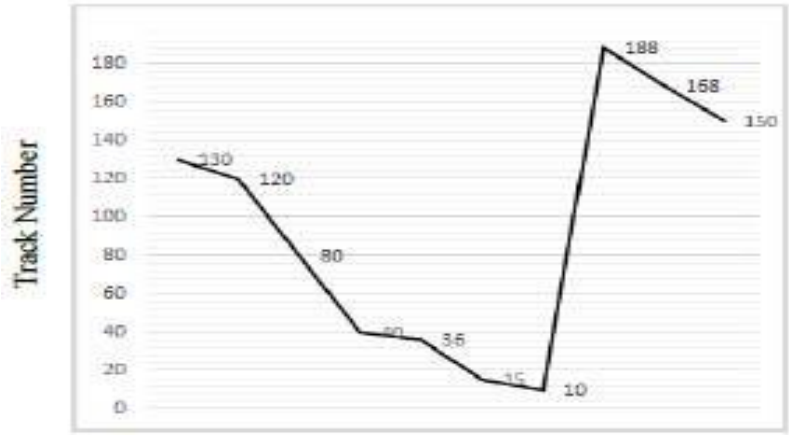


Figure5.2.6: C-LOOK[c]

Total head movement = |130-120| + |120-80| + |80-40| + |40-36| + |36-15| + |15-10| + |188-168| + |168-150| = 158

Average Seek Time = 158 / 9 ms = 17.55 ms.

## 5.3. Similarly all the calculations have been performed in the following test cases:

### 5.3.1. Case II:

Suppose the head of a moving – head disk with 181 tracks numbered 0 to 180) is currently serving request at tracks 45. The queue it requests is kept in the FIFO order: 25, 10, 151, 170, 62, 46, 74, 111.

### 5.3.2. Case III:

Suppose the head of a moving – head disk with 100 tracks numbered 0 to 99) is currently serving request at tracks 63. The queue it requests is kept in the FIFO order: 33, 72, 47, 8, 99, 74, 52, 75.

### 5.3.3. Case IV:

Suppose the head of a moving – head disk with 5000 tracks numbered 0 to 4999) is currently serving request at tracks 143. The queue it requests is kept in the FIFO order: 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.

### 5.3.4. Case V:

Suppose the head of a moving – head disk with 201 tracks numbered 0 to 200) is currently serving request at tracks 100. The queue it requests is kept in the FIFO order: 30, 86, 90, 100, 105, 110, 135, 145.

# CHAPTER-6
# Architectural design of the project

## 6.1. DSA ALGORITHM:

The main aim of our proposed DSA algorithm is to improve the disk performance by reducing average seek time of the disk scheduling algorithm. So that there will be faster data transfer. The main goal behind all is to enhance the system performance.

### 6.1.1. Proposed DSA Algorithm:

we calculate the average seek time and the transfer time. We calculate the transfer time by using the following formula shown:

$$Ta = Ts + ( 1/2R ) + ( B/RN )$$

- Ta = Transfer Time
- Ts = Average Seek Time
- B = Number of bytes to be transferred
- N = Number of bytes on track
- R = Rotation speed in revolutions per second

## 6.2. Pseudo code:

```
if (DQ != NULL)
        //A Disk Queue with requests for accessing tracks
Read IDHP
        //All the TRs present in DQ are sorted in ascending order (numerical order)
if (| IDHP - LTR |) < (| IDHP - HTR |)
        {
                IST = | IDHP - LTR |
        }
        //Scanning will start from the LTR
         else if
                 (| IDHP - LTR |) > (| IDHP - HTR |)
        {
                IST = | IDHP - HTR |
        }
        //Scanning will start from the HTR
        Else
        {
        //Scanning can be done from any of the end
        IST = | IDHP - LTR | or| IDHP -HTR |
        }
```

```
        end if
ST ← 0
    // initializing ST to 0
for  i = 1 to n
    ST = ST + | TRi+1 - TRi | end for
    ST = ST + IST
     //Calculate AST and TT
```

### 6.2.1. Explain For The Terms:

- DQ = a Disk Queue
- IDHP = Initial Disk Head Position
- TR = Track Request
- n = number of TRs
- LTR = Lowest Track Request
- HTR = Highest Track Request
- ST = Seek Time
- i = loop variable
- AST = Average Seek Time
- TT = Transfer Time

## 6.3. Flowchart Explanation:

In our proposed DSA algorithm, the requests in the disk queue are to be sorted according to the track number requested. Then we calculate the absolute difference between the initial disk head position (IDHP) and the lowest track request (LTR) of disk queue and absolute difference of the initial disk head position (IDHP) and the highest track request (HTR)of the disk queue. If (|IDHP – LTR|) is greater than (|IDHP – HTR|), then we scan the requests in ascending order starting from the initial position and if (|IDHP – LTR|) is less than (|IDHP – HTR|), then scanning starts in descending order (Highest track number to lowest track number). If (|IDHP – LTR|) is equal to (|IDHP – HTR|), then scanning can start from any of the side.

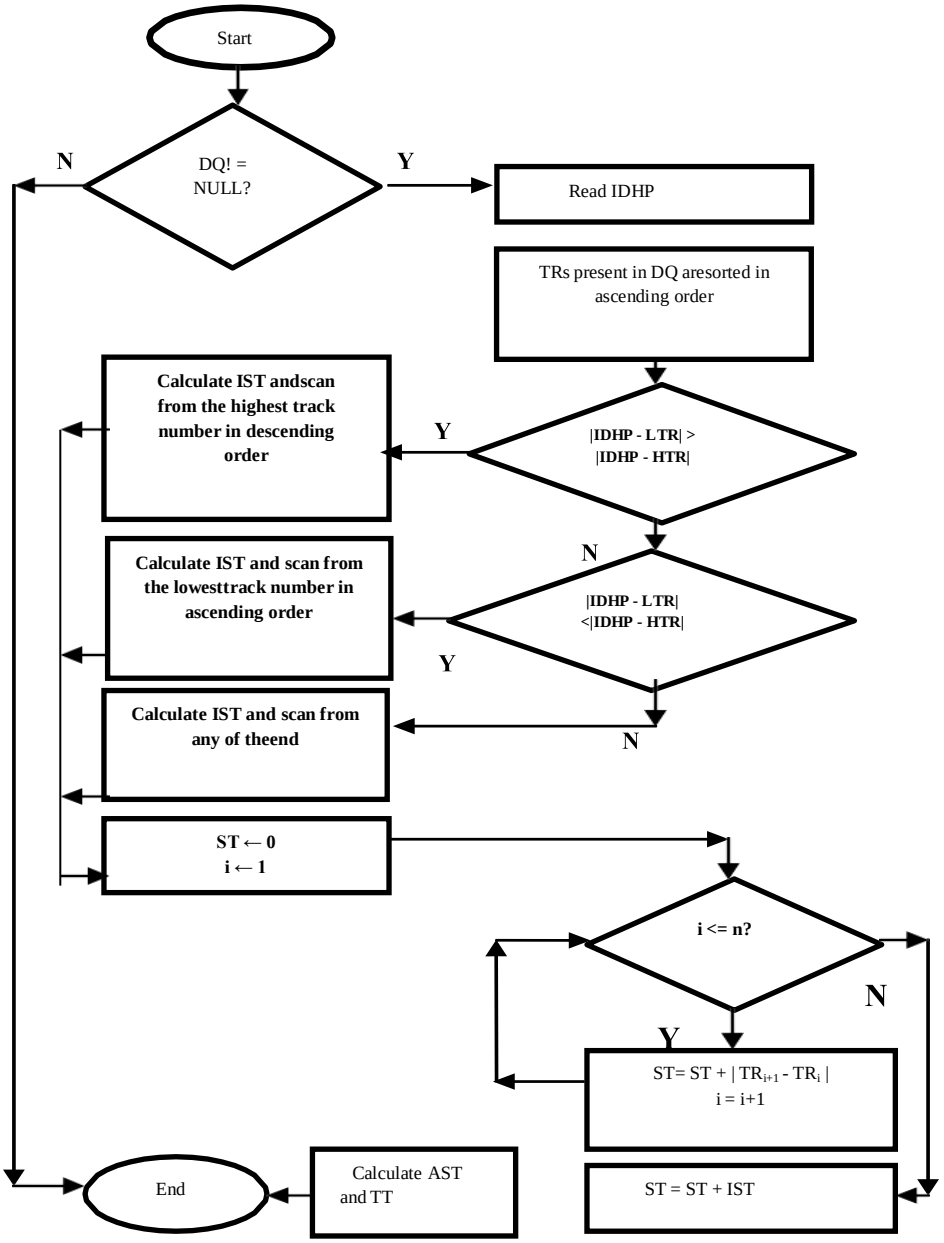


Figure6.3:Flowchart Of DSA Algorithm

# CHAPTER-7
# System design-Modules

## 7.1. System Context Diagram:

A Context Diagram is the highest level of data flow diagram. It represents the flows of information between the system and all external entities that may have interact with a system. The entire software system is shown as a single process.

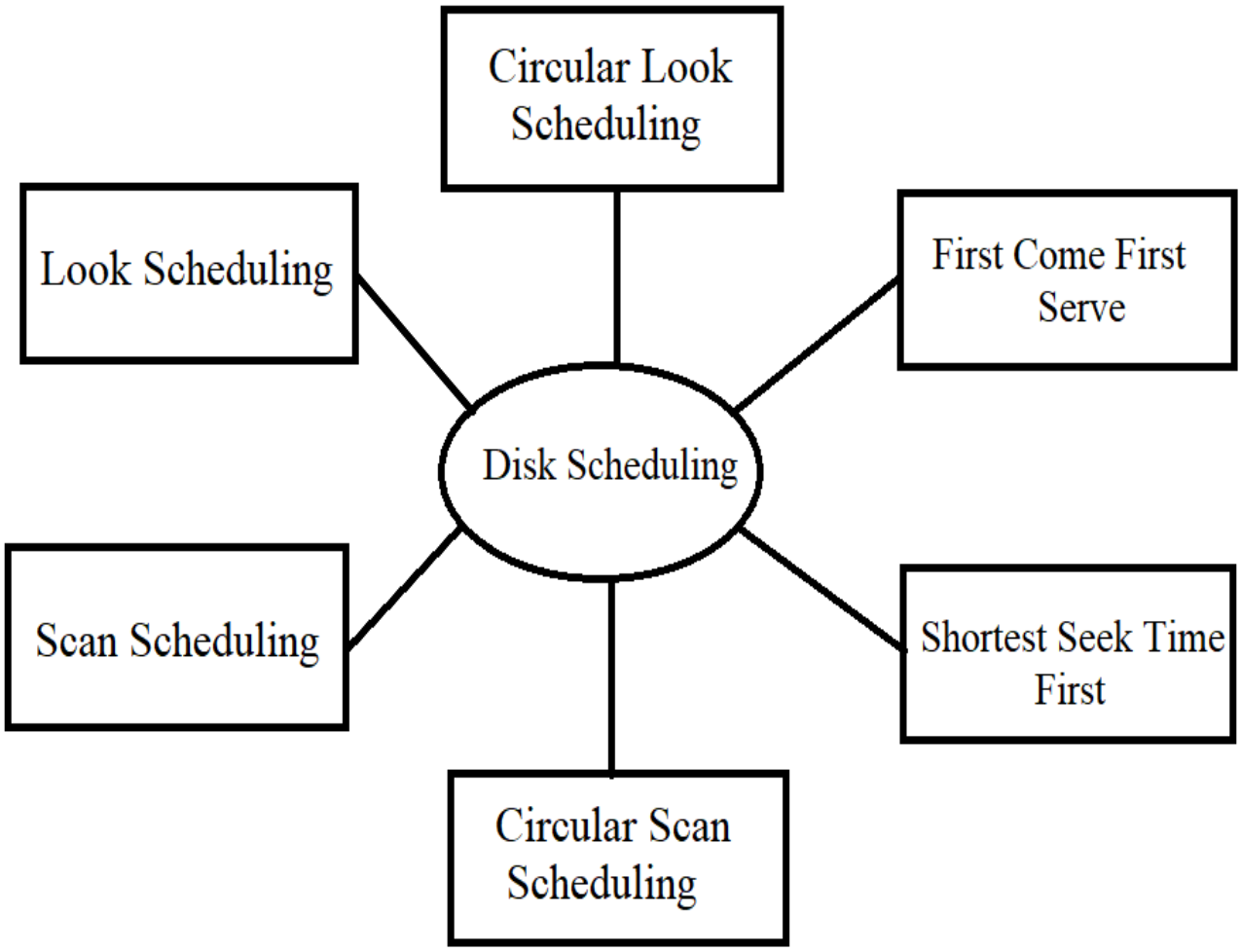


Figure7.1: Context Diagram For Disk Scheduling

# CHAPTER-8
# Testing

## 8.1. Testing & Explanation:

In the stage of implementation of the research project, the researcher gives the clear picture of the research task and the process through which the researcher sought to reach the goal of the research. The researcher has used experimental research methodology in this research. The researcher has completed the literature review and prepared the sample I/O request queue on the basis of the literature review. At the initial phase, 5 requests are selected and manually calculated the Total head movement and Average seek time.

The researcher has formulated the five sample I/O requests having different cylinder size with different cases and have calculated total head movement and average seek time of six different algorithms (FCFS, SSTF, SCAN, C-SCAN, LOOK and C-LOOK) and finally have calculated the average seek time of all algorithms.

We compares the average head movement of six disk scheduling algorithms for the first 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** |

Figure8.1.1: Average head movement of first five runs and their average.

Shortest Seek Time First algorithm produces the minimum head movement of 240 in thefirst 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 very much clear that forthe different five runs, the Shortest Seek Time First disk scheduling algorithm has minimum totalhead movement in all cases as compared to other five algorithms. A graphical representation is shown in the figure 8 with the help of data evaluated in table.

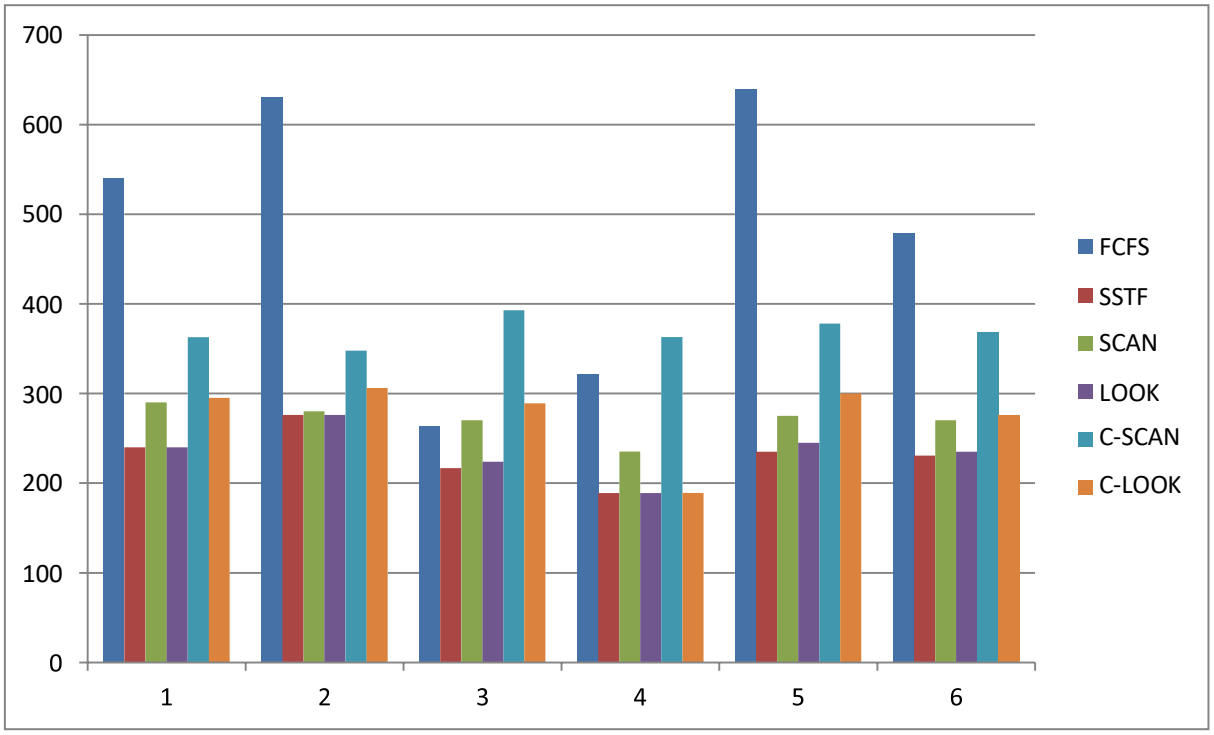


Figure 8.1.2: Comparison between six disk scheduling algorithms in graphical representation.

The figure 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 average total headmovement of each algorithm.

From the various runs, it is concluded that Shortest Seek Time First has the minimum average head movement than all other five algorithms for the similar requests. Thus Shortest Seek Time First algorithm is a good criterion for selecting the requestsfor I/O from the disk queue.

To evaluate the performance of our proposed algorithm, we have taken three different cases. In each case, we have compared the experimental results of our proposed algorithm with other disk scheduling algorithms.

We have taken the following track requests for accessing the tracks as (25, 10, 151, 170, 62, 46, 74 and 111) and the initial disk head position is at 45. Table 1 shows the comparison of all the algorithms with our proposed algorithm. Figure 3, Figure 4, Figure 5, Figure 6, Figure 7 and Figure 8 shows the representation of FIFO, SSTF, SCAN, C-SCAN, LOOK and DSA respectively. Figure 9 and Figure 10 shows the comparison of average seek time and transfer time respectively.

| Algorithms | Average Seek Time | Transfer Time |
|---|---|---|
| FIFO | 48 | 48.01191 |
| SSTF | 35.625 | 35.63691 |
| SCAN | 38.125 | 38.13691 |
| C-SCAN | 42.5 | 42.51191 |
| LOOK | 37.5 | 37.51191 |

Figure8.1.3: Comparison of all algorithms

## 8.2. Sample Disk Queue:

| Case No. | Track range | Queue | Initial head position |
|---|---|---|---|
| 1 | 0-199 | 98, 183, 37, 122, 14, 124, 65, 67 [18] | 53 |
| 2 | 0-99 | 33, 72, 47, 8, 99, 74, 52, 75 [18] | 63 |
| 3 | 0-1999 | 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 [18] | 143 |
| 4 | 0-180 | 25, 10, 151, 170, 62, 46, 74, 111 [16] | 45 |
| 5 | 0-180 | 16, 75, 24, 21, 30, 80, 116, 63 [16] | 66 |
| 6 | 0-180 | 25, 33, 54, 64, 40, 90, 110, 160 [16] | 125 |
| 7 | 0-199 | 95, 180, 34, 119, 11, 123, 62, 64 [19] | 50 |
| 8 | 0-199 | 91, 150, 42, 130, 18, 140, 70, 60 [9] | 50 |
| 9 | 0-39 | 1, 36, 16, 34, 9, 12, 8, 13 [12] | 11 |
| 10 | 0-99 | 2, 5, 7, 10, 15, 17, 19, 23, 26, 31, 35 [20] | 53 |
| 11 | 0-199 | 47, 35, 180, 132, 156, 23, 34, 5, 210, 83, 96, 103, 88, 76, 113, 128, 73, 120, 169, 123, 111, 179, 136, 40 [20] | 53 |
| 12 | 0-199 | 100, 180, 40, 120, 20, 130, 60, 70 [8] | 59 |
| 13 | 0-99 | 70, 60, 80, 40, 10, 15 [7] | 20 |
| 14 | 0-99 | 16, 70, 20, 98, 45, 5 [7] | 60 |
| 15 | 0-99 | 10, 97, 17, 65, 70, 12, 8, 15, 90, 5 [7] | 26 |
| 16 | 0-199 | 6, 143, 15, 185, 85, 120, 33, 28, 146 [1] | 70 |
| 17 | 0-199 | 113, 190, 45, 135, 20, 156, 65, 75 [21] | 53 |
| 18 | 0-199 | 36, 180, 120, 10, 15, 40, 188, 150, 168 [13] | 130 |
| 19 | 0-200 | 38, 180, 130, 10, 50, 15, 190, 90, 150 [5] | 120 |
| 20 | 0-199 | 25, 90, 135, 50, 190, 60 [2] | 100 |

Figure 8.2: Sample Queue Values

## 8.3. Comparisons:

Following figure shows the comparisons of the average value of Total Head Movement (THM) and Average Seek Time (AST) of all the algorithms for different cases. The above figure shows the comparative analysis of different algorithms with different test cases. The average THM of FCFS algorithm is found that 841.5 ms and average AST is found as 94.84 ms which is the highest value among the other algorithms.

Similarly the average THM and average AST of SSTF algorithm is found as 280.75 and 33.06 ms respectively. Which is the improved value than FCFS algorithm. Again, we can observed that the average THM and average AST of SCAN algorithm is 289.70ms and 34.20ms respectively which is little more than SSTF algorithm.

The average THM and average AST of C- SCAN algorithm is 198.75ms and 23.39ms respectively which is lower than FCFS, SSTF and SCAN algorithm. The average THM and average AST of LOOK algorithm is 259.20ms and 30.45ms which is higher than C-SCAN but lower than FCFS, SSTF and SCAN algorithms. The lowest value of average THM and average AST is recorded on the C-LOOK algorithm which is 157.55ms and 18.46ms respectively.

Hence, by the comparative analysis of the above figure the THM and AST of C-LOOK algorithm is improvised than other remaining algorithms.

### 8.3.1. FCFS Scheduling:

This algorithm is very simple and easy to understand and implement. Each and every process gets a chance to execute, so no starvation occurs. It is suitable for the disk having light load. It gives very low throughput due to the lengthy seeks.

### 8.3.2. SSTF Scheduling:

The average seek time is reduced compared to FCFS. So, this algorithm gives higher throughput compared to FCFS scheduling algorithm. It is suitable for the batch processing system. There is a chance of starvation in this algorithm so that it does not ensure fairness and this algorithm leads to higher variances of response times.

### 8.3.3. SCAN Scheduling:

It offers an improved variance of response time than previous algorithms. The major drawback on this algorithm is, it does not change its direction until the end of the disk is reached despite the absence of requests to be serviced. So, it is recommended for light load.

### 8.3.4. C-SCAN Scheduling:

It is the successor and the improved version of the SCAN algorithm so, this algorithm limits the variance of response time by avoiding discrimination against the innermost and outermost cylinders so that high level of throughput is maintained.

### 8.3.5. LOOK Scheduling:

This algorithm performs sweeps large enough to service all requests which is the major advantage of this algorithm and it improves efficiency by avoiding unnecessary seek operation so that throughput is increased.

## 8.3.6. C-LOOK Scheduling:

In this algorithm the head does not have to move till the end of the disk if there are no requests to be serviced. It gives lower variance of response time than LOOK and starvation is avoided. Thus from this broad comparison of different algorithms, we found that C-LOOK algorithm will result

| Case No. | FCFS | | SSTF | | SCAN | | C-SCAN | | LOOK | | C-LOOK | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | THM | AST | THM | AST | THM | AST | THM | AST | THM | AST | THM | AST |
| 1 | 640 | 80.00 | 236 | 29.50 | 236 | 29.50 | 187 | 23.38 | 208 | 26.00 | 157 | 19.63 |
| 2 | 294 | 36.75 | 170 | 21.25 | 162 | 20.25 | 90 | 11.25 | 146 | 18.25 | 82 | 10.25 |
| 3 | 7081 | 786.78 | 1745 | 193.89 | 1917 | 213.0 | 1229 | 136.56 | 1745 | 193.89 | 918 | 102.0 |
| 4 | 384 | 48.00 | 285 | 35.63 | 215 | 26.88 | 179 | 22.38 | 195 | 24.38 | 159 | 19.88 |
| 5 | 311 | 38.88 | 156 | 19.50 | 182 | 22.75 | 171 | 21.38 | 150 | 18.75 | 91 | 11.38 |
| 6 | 283 | 35.38 | 235 | 29.38 | 285 | 35.63 | 145 | 18.13 | 235 | 29.38 | 100 | 12.50 |
| 7 | 644 | 80.50 | 236 | 29.50 | 230 | 28.75 | 187 | 23.38 | 208 | 26.00 | 157 | 19.63 |
| 8 | 610 | 76.25 | 248 | 31.00 | 200 | 25.00 | 189 | 23.63 | 164 | 20.50 | 122 | 15.25 |
| 9 | 120 | 15.00 | 55 | 6.88 | 47 | 5.88 | 38 | 4.75 | 45 | 5.63 | 34 | 4.25 |
| 10 | 84 | 7.64 | 51 | 4.64 | 51 | 4.64 | 51 | 4.64 | 51 | 4.64 | 51 | 4.64 |
| 11 | 1255 | 52.29 | 253 | 10.54 | 233 | 9.71 | 179 | 7.46 | 223 | 9.29 | 155 | 6.46 |
| 12 | 631 | 78.88 | 221 | 27.63 | 239 | 29.88 | 198 | 24.75 | 199 | 24.88 | 159 | 19.88 |
| 13 | 155 | 25.83 | 80 | 13.33 | 100 | 16.67 | 79 | 13.17 | 80 | 13.33 | 50 | 8.33 |
| 14 | 319 | 53.17 | 168 | 28.00 | 158 | 26.33 | 89 | 14.83 | 148 | 24.67 | 83 | 13.83 |
| 15 | 465 | 46.50 | 113 | 11.30 | 123 | 12.30 | 60 | 6.00 | 113 | 11.30 | 53 | 5.30 |
| 16 | 844 | 93.78 | 294 | 32.67 | 255 | 28.33 | 184 | 20.44 | 243 | 27.00 | 164 | 18.22 |
| 17 | 724 | 90.50 | 323 | 40.38 | 243 | 30.38 | 187 | 23.38 | 203 | 25.38 | 158 | 19.75 |
| 18 | 642 | 71.33 | 256 | 28.44 | 318 | 35.33 | 179 | 19.89 | 298 | 33.11 | 158 | 17.56 |
| 19 | 804 | 89.33 | 250 | 27.78 | 310 | 34.44 | 190 | 21.11 | 290 | 32.22 | 170 | 18.89 |
| 20 | 540 | 90.00 | 240 | 40.00 | 290 | 48.33 | 164 | 27.33 | 240 | 40.00 | 130 | 21.67 |
| Average | 841.5 | 94.84 | 280.75 | 33.06 | 289.7 | 34.20 | 198.75 | 23.39 | 259.2 | 30.43 | 157.55 | 18.46 |

Figure 8.3.6: Final Result For All Algorithm

From the above comparison of the twenty different cases with the implementation of the different scheduling algorithms i.e. FCFS, SSTF, SCAN, C- SCAN, LOOK, C-LOOK using the simulator made in Python & Java Programming Language, we have obtained the Total Head Movement (THM) and Average Seek Time (AST). After analysis all the result it is found that average seek time of C-LOOK algorithm have minimum value than other algorithms

# CHAPTER-9
# Appendices

## 9.1. Source Code:

### 9.1.1. LaunchFrame.java

```
package DiskScheduling;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.HeadlessException;
import java.awt.Image;
import java.awt.Toolkit;
import static java.lang.Math.abs;
import javax.accessibility.AccessibleContext;
import javax.swing.JOptionPane;
import java.io.*;
import javax.swing.ImageIcon;

public class launchFrame extends javax.swing.JFrame {

    Color defaultBackground = new Color(0,153,153);
    Color defaultForeground = new Color(0,51,51);
    int ctr = 0;
    static int ARRAY[] = new int[1000];
    static int SORTED_ARRAY[] = new int[1000];
    static int returnArray[] = new int[1000];
    static int END = 0;
    int THM = 0;
    static String DISPLAY;
    int x;
    float ST = 0;
    static int DIRECTION = 0;
    int temp_END = 0;


    public launchFrame() {
        SplashScreen splash = new SplashScreen(1000);
        splash.showSplashAndExit();
        for( x = 0;x<1000;x++)
        {
            ARRAY[x] = '\0';
            SORTED_ARRAY[x] = '\0';
        }
        ImageIcon icon = new ImageIcon("rsc\\disk_scheduling.png");
        setIconImage(icon.getImage());
        initComponents();
        fcfsBttn.setSelected(true);
        this.setLocationRelativeTo(null);
    }


    private void initComponents() {

        buttonGroup1 = new javax.swing.ButtonGroup();
        jMenu3 = new javax.swing.JMenu();
        panel = new javax.swing.JPanel();
        jPanel1 = new javax.swing.JPanel();
        exitBttn = new javax.swing.JButton();
```

```
        clearBttn = new javax.swing.JButton();
        diskSchedLbl = new javax.swing.JLabel();
        design1 = new javax.swing.JLabel();
        design2 = new javax.swing.JLabel();
        scanTypePanel = new javax.swing.JPanel();
        computeBttn = new javax.swing.JButton();
        fcfsBttn = new javax.swing.JRadioButton();
        sstfBttn = new javax.swing.JRadioButton();
        scanBttn = new javax.swing.JRadioButton();
        cScanBttn = new javax.swing.JRadioButton();
        lookBttn = new javax.swing.JRadioButton();
        scanTypeLbl = new javax.swing.JLabel();
        cLookBttn = new javax.swing.JRadioButton();
        infoPanel = new javax.swing.JPanel();
        trackNoLbl = new javax.swing.JLabel();
        CurrPosLbl = new javax.swing.JLabel();
        PrevPosLbl = new javax.swing.JLabel();
        dirLbl = new javax.swing.JLabel();
        seekRateLbl = new javax.swing.JLabel();
        trackNoField = new javax.swing.JTextField();
        currPosField = new javax.swing.JTextField();
        editLbl = new javax.swing.JLabel();
        prevPosBttn = new javax.swing.JToggleButton();
        valOfAField = new javax.swing.JTextField();
        StatusPanel = new javax.swing.JPanel();
        diagramScrollPane = new javax.swing.JScrollPane();
        statusArea = new javax.swing.JTextArea();
        showAreaBttn = new javax.swing.JLabel();
        showSortedArrayBttn = new javax.swing.JLabel();
        clearStatusAreaBttn = new javax.swing.JLabel();

        jMenu3.setText("jMenu3");

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Disk Scheduling Algorithm");
        setCursor(new java.awt.Cursor(java.awt.Cursor.HAND_CURSOR));
        setForeground(new java.awt.Color(255, 255, 255));
        setIconImages(null);
        setLocation(new java.awt.Point(50, 50));
        setName("Disk Scheduling Algorithm"); // NOI18N
        setUndecorated(true);
        setResizable(false);

        panel.setBackground(new java.awt.Color(0, 102, 102));
        panel.setBorder(javax.swing.BorderFactory.createEtchedBorder());

        jPanel1.setBackground(new java.awt.Color(0, 51, 51));

jPanel1.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.border.BevelBorder
.RAISED));

        exitBttn.setBackground(new java.awt.Color(0, 102, 102));
        exitBttn.setFont(diskSchedLbl.getFont());
        exitBttn.setForeground(new java.awt.Color(255, 255, 255));
        exitBttn.setText("Exit");
```

```
        exitBttn.setBorder(new   javax.swing.border.LineBorder(new   java.awt.Color(51,   255,
255), 1, true));
        exitBttn.setContentAreaFilled(false);
        exitBttn.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        exitBttn.setDefaultCapable(false);
        exitBttn.setFocusPainted(false);
        exitBttn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                exitBttnActionPerformed(evt);
            }
        });

        clearBttn.setBackground(new java.awt.Color(0, 153, 153));
        clearBttn.setFont(diskSchedLbl.getFont());
        clearBttn.setForeground(new java.awt.Color(255, 255, 255));
        clearBttn.setText("Clear");
        clearBttn.setBorder(new   javax.swing.border.LineBorder(new   java.awt.Color(51,   255,
255), 1, true));
        clearBttn.setContentAreaFilled(false);
        clearBttn.setEnabled(false);
        clearBttn.setRequestFocusEnabled(false);
        clearBttn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                clearBttnActionPerformed(evt);
            }
        });

        diskSchedLbl.setFont(new java.awt.Font("Times New Roman", 1, 14)); // NOI18N
        diskSchedLbl.setForeground(new java.awt.Color(255, 255, 255));
        diskSchedLbl.setText("Disk Scheduling Algorithm");

        design1.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/DiskScheduling/socbiz-loader.gif")));       //
NOI18N
        design1.setText("jLabel2");

        design2.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/DiskScheduling/socbiz-loader.gif")));       //
NOI18N
        design2.setText("jLabel2");

        scanTypePanel.setBackground(new java.awt.Color(0, 102, 102));
        scanTypePanel.setBorder(javax.swing.BorderFactory.createEtchedBorder());

        computeBttn.setBackground(new java.awt.Color(255, 255, 255));
        computeBttn.setFont(diskSchedLbl.getFont());
        computeBttn.setForeground(new java.awt.Color(0, 51, 51));
        computeBttn.setText("Compute");
        computeBttn.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        computeBttn.setBorderPainted(false);
        computeBttn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                computeBttnActionPerformed(evt);
            }
        });
```

```
fcfsBttn.setBackground(new java.awt.Color(0, 102, 102));
buttonGroup1.add(fcfsBttn);
fcfsBttn.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
fcfsBttn.setForeground(new java.awt.Color(255, 255, 255));
fcfsBttn.setText("First Come First Serve");
fcfsBttn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        fcfsBttnActionPerformed(evt);
    }
});

sstfBttn.setBackground(new java.awt.Color(0, 102, 102));
buttonGroup1.add(sstfBttn);
sstfBttn.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
sstfBttn.setForeground(new java.awt.Color(255, 255, 255));
sstfBttn.setText("Shortest Seek Time First");
sstfBttn.setActionCommand("");
sstfBttn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        sstfBttnActionPerformed(evt);
    }
});

scanBttn.setBackground(new java.awt.Color(0, 102, 102));
buttonGroup1.add(scanBttn);
scanBttn.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
scanBttn.setForeground(new java.awt.Color(255, 255, 255));
scanBttn.setText("SCAN");
scanBttn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        scanBttnActionPerformed(evt);
    }
});

cScanBttn.setBackground(new java.awt.Color(0, 102, 102));
buttonGroup1.add(cScanBttn);
cScanBttn.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
cScanBttn.setForeground(new java.awt.Color(255, 255, 255));
cScanBttn.setText("C-SCAN");
cScanBttn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        cScanBttnActionPerformed(evt);
    }
});

lookBttn.setBackground(new java.awt.Color(0, 102, 102));
buttonGroup1.add(lookBttn);
lookBttn.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
lookBttn.setForeground(new java.awt.Color(255, 255, 255));
lookBttn.setText("LOOK");
lookBttn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        lookBttnActionPerformed(evt);
    }
});
```

```
        scanTypeLbl.setFont(new java.awt.Font("Times New Roman", 1, 36)); // NOI18N
        scanTypeLbl.setForeground(new java.awt.Color(255, 255, 255));
        scanTypeLbl.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        scanTypeLbl.setText("Scan Type");
        scanTypeLbl.setBorder(javax.swing.BorderFactory.createEtchedBorder());

        cLookBttn.setBackground(new java.awt.Color(0, 102, 102));
        buttonGroup1.add(cLookBttn);
        cLookBttn.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
        cLookBttn.setForeground(new java.awt.Color(255, 255, 255));
        cLookBttn.setText("C-LOOK");
        cLookBttn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                cLookBttnActionPerformed(evt);
            }
        });

        infoPanel.setBackground(new java.awt.Color(0, 102, 102));
        infoPanel.setBorder(javax.swing.BorderFactory.createEtchedBorder());

        trackNoLbl.setFont(new java.awt.Font("Times New Roman", 0, 24)); // NOI18N
        trackNoLbl.setForeground(new java.awt.Color(255, 255, 255));
        trackNoLbl.setText("No. of tracks:");
        trackNoLbl.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

        CurrPosLbl.setFont(trackNoLbl.getFont());
        CurrPosLbl.setForeground(new java.awt.Color(255, 255, 255));
        CurrPosLbl.setText("Current Position:");
        CurrPosLbl.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

        PrevPosLbl.setFont(trackNoLbl.getFont());
        PrevPosLbl.setForeground(new java.awt.Color(255, 255, 255));
        PrevPosLbl.setText("Previous Position:");
        PrevPosLbl.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

        dirLbl.setFont(trackNoLbl.getFont());
        dirLbl.setForeground(new java.awt.Color(255, 255, 255));
        dirLbl.setText("Direction:");
        dirLbl.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

        seekRateLbl.setFont(trackNoLbl.getFont());
        seekRateLbl.setForeground(new java.awt.Color(255, 255, 255));
        seekRateLbl.setText("Seek Rate (ms):");
        seekRateLbl.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);

        trackNoField.setBackground(new java.awt.Color(0, 102, 102));
        trackNoField.setFont(trackNoLbl.getFont());
        trackNoField.setForeground(new java.awt.Color(255, 255, 255));
        trackNoField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        trackNoField.setText("0");
        trackNoField.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(51, 255,
255), 1, true));
        trackNoField.setCursor(new java.awt.Cursor(java.awt.Cursor.TEXT_CURSOR));
        trackNoField.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyReleased(java.awt.event.KeyEvent evt) {
                trackNoFieldKeyReleased(evt);
```

```
            }
        });

        currPosField.setBackground(new java.awt.Color(0, 102, 102));
        currPosField.setFont(trackNoLbl.getFont());
        currPosField.setForeground(new java.awt.Color(255, 255, 255));
        currPosField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        currPosField.setText("0");
        currPosField.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(51, 255,
255), 1, true));
        currPosField.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyReleased(java.awt.event.KeyEvent evt) {
                currPosFieldKeyReleased(evt);
            }
        });

        prevPosField.setBackground(new java.awt.Color(0, 102, 102));
        prevPosField.setFont(trackNoLbl.getFont());
        prevPosField.setForeground(new java.awt.Color(255, 255, 255));
        prevPosField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        prevPosField.setText("0");
        prevPosField.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(51, 255,
255), 1, true));
        prevPosField.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyReleased(java.awt.event.KeyEvent evt) {
                prevPosFieldKeyReleased(evt);
            }
        });

        seekRateField.setBackground(new java.awt.Color(0, 102, 102));
        seekRateField.setFont(trackNoLbl.getFont());
        seekRateField.setForeground(new java.awt.Color(255, 255, 255));
        seekRateField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        seekRateField.setText("0");
        seekRateField.setBorder(new    javax.swing.border.LineBorder(new    java.awt.Color(51,
255, 255), 1, true));
        seekRateField.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                seekRateFieldActionPerformed(evt);
            }
        });

        nextBttn.setBackground(new java.awt.Color(255, 255, 255));
        nextBttn.setFont(diskSchedLbl.getFont());
        nextBttn.setForeground(new java.awt.Color(0, 51, 51));
        nextBttn.setText("ADD");
        nextBttn.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        nextBttn.setBorderPainted(false);
        nextBttn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                nextBttnActionPerformed(evt);
            }
        });

        endBttn.setBackground(new java.awt.Color(255, 255, 255));
        endBttn.setFont(diskSchedLbl.getFont());
```

```
        endBttn.setForeground(new java.awt.Color(0, 51, 51));
        endBttn.setText("END");
        endBttn.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        endBttn.setBorderPainted(false);
        endBttn.setEnabled(false);
        endBttn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                endBttnActionPerformed(evt);
            }
        });

        editBttn.setBackground(new java.awt.Color(255, 255, 255));
        editBttn.setFont(diskSchedLbl.getFont());
        editBttn.setForeground(new java.awt.Color(0, 51, 51));
        editBttn.setText("EDIT");
        editBttn.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        editBttn.setBorderPainted(false);
        editBttn.setEnabled(false);
        editBttn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                editBttnActionPerformed(evt);
            }
        });

        editField.setBackground(new java.awt.Color(0, 102, 102));
        editField.setForeground(new java.awt.Color(255, 255, 255));
        editField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        editField.setText("0");
        editField.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Replacement
No.", javax.swing.border.TitledBorder.CENTER, javax.swing.border.TitledBorder.TOP, new
java.awt.Font("Tahoma", 0, 11), new java.awt.Color(255, 255, 255))); // NOI18N
        editField.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                editFieldActionPerformed(evt);
            }
        });

        indexField.setBackground(new java.awt.Color(0, 102, 102));
        indexField.setForeground(new java.awt.Color(255, 255, 255));
        indexField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        indexField.setText("0");
        indexField.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Index of
Array", javax.swing.border.TitledBorder.CENTER, javax.swing.border.TitledBorder.TOP, new
java.awt.Font("Tahoma", 0, 11), new java.awt.Color(255, 255, 255))); // NOI18N

        editLbl.setFont(trackNoLbl.getFont());
        editLbl.setForeground(new java.awt.Color(255, 255, 255));
        editLbl.setText("Edit:");

        prevPosBttn.setBackground(new java.awt.Color(255, 255, 255));
        prevPosBttn.setFont(new java.awt.Font("Times New Roman", 1, 24)); // NOI18N
        prevPosBttn.setText("ON");
        prevPosBttn.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        prevPosBttn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                prevPosBttnActionPerformed(evt);
```

```
            }
        });

        javax.swing.GroupLayout infoPanelLayout = new javax.swing.GroupLayout(infoPanel);
        infoPanel.setLayout(infoPanelLayout);
        infoPanelLayout.setHorizontalGroup(
            infoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(infoPanelLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(infoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
infoPanelLayout.createSequentialGroup()

.addGroup(infoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                            .addGroup(infoPanelLayout.createSequentialGroup()
                                .addComponent(CurrPosLbl,
javax.swing.GroupLayout.DEFAULT_SIZE,
.addGroup(infoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(seekRateLbl,   javax.swing.GroupLayout.PREFERRED_SIZE,   31,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(seekRateField, javax.swing.GroupLayout.PREFERRED_SIZE, 52,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(15, 15, 15)

.addGroup(infoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(arrayField,   javax.swing.GroupLayout.PREFERRED_SIZE,   53,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup(infoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(endBttn,   javax.swing.GroupLayout.PREFERRED_SIZE,   53,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addGroup(infoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(infoPanelLayout.createSequentialGroup()
                        .addGap(17, 17, 17)
                        .addComponent(editLbl)
                        .addGap(0, 14, Short.MAX_VALUE))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
infoPanelLayout.createSequentialGroup()
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(editBttn,     javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))))
                .addContainerGap())
        );
        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(design1,       javax.swing.GroupLayout.PREFERRED_SIZE,       57,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
```

```
                .addComponent(design2,      javax.swing.GroupLayout.PREFERRED_SIZE,      60,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(240, 240, 240)
                .addComponent(diskSchedLbl)
                .addGap(222, 222, 222)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(exitBttn,    javax.swing.GroupLayout.PREFERRED_SIZE,    109,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(clearBttn,   javax.swing.GroupLayout.PREFERRED_SIZE,   109,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
        showSortedArrayBttn.setFont(new java.awt.Font("Times New Roman", 1, 24)); // NOI18N
        showSortedArrayBttn.setForeground(new java.awt.Color(255, 255, 255));
        showSortedArrayBttn.setText("SA");
        showSortedArrayBttn.setToolTipText("Show Sorted Array if available");
        showSortedArrayBttn.setBorder(new                    javax.swing.border.LineBorder(new
java.awt.Color(0, 204, 204), 1, true));
        showSortedArrayBttn.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                showSortedArrayBttnMouseClicked(evt);
            }
        });

        clearStatusAreaBttn.setFont(new java.awt.Font("Times New Roman", 1, 24)); // NOI18N
        clearStatusAreaBttn.setForeground(new java.awt.Color(255, 255, 255));
        clearStatusAreaBttn.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        clearStatusAreaBttn.setText("C");
        clearStatusAreaBttn.setToolTipText("Clear");
        clearStatusAreaBttn.setBorder(new                    javax.swing.border.LineBorder(new
java.awt.Color(0, 255, 255), 1, true));
        clearStatusAreaBttn.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                clearStatusAreaBttnMouseClicked(evt);
            }
        });
        javax.swing.GroupLayout panelLayout = new javax.swing.GroupLayout(panel);
        panel.setLayout(panelLayout);
        panelLayout.setHorizontalGroup(
            panelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1,                      javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(panelLayout.createSequentialGroup()
                .addComponent(StatusPanel,            javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(0, 0, Short.MAX_VALUE))
        );
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(panel,         javax.swing.GroupLayout.PREFERRED_SIZE,         895,
javax.swing.GroupLayout.PREFERRED_SIZE)
        );
        layout.setVerticalGroup(
```

```
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(panel,                    javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void cLookBttnActionPerformed(java.awt.event.ActionEvent evt) {
        valOfAField.setEnabled(true);
    }

    private void lookBttnActionPerformed(java.awt.event.ActionEvent evt){
        valOfAField.setEnabled(false);
    }

    private void cScanBttnActionPerformed(java.awt.event.ActionEvent evt) {
        valOfAField.setEnabled(true);
    }

    private void scanBttnActionPerformed(java.awt.event.ActionEvent evt) {
        valOfAField.setEnabled(false);
    }

    private void sstfBttnActionPerformed(java.awt.event.ActionEvent evt){
        valOfAField.setEnabled(false);
    }

    private void fcfsBttnActionPerformed(java.awt.event.ActionEvent evt) {
        valOfAField.setEnabled(false);
    }

    private void computeBttnActionPerformed(java.awt.event.ActionEvent evt) {

        statusArea.setText(statusArea.getText()+"\nComputing . . .");
        END = temp_END;
        try
        {
            if(fcfsBttn.isSelected() == true)
                fcfsProcess();
            else
            {
                sortArray();
                if(cScanBttn.isSelected()==true)
                c_scanProcess();

                if(scanBttn.isSelected() == true)
                scanProcess();

                if(lookBttn.isSelected()==true)
                lookProcess();

                if(cLookBttn.isSelected()==true)
                c_lookProcess();
```

```
                if(sstfBttn.isSelected()==true)
                sstfProcess();
            }

            ResultingWindow r = new ResultingWindow();
            statusArea.setText(statusArea.getText()+"\n"+DISPLAY);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }

    }//GEN-LAST:event_computeBttnActionPerformed

    private    void    clearBttnActionPerformed(java.awt.event.ActionEvent    evt)    {//GEN-
FIRST:event_clearBttnActionPerformed
        trackNoField.setText("0");
        currPosField.setText("0");
        prevPosField.setText("0");
        seekRateField.setText("0");
        arrayField.setText("");
        indexField.setText("0");
        editField.setText("0");
        statusArea.setText("");
        nextBttn.setEnabled(true);
        arrayField.setEnabled(true);

        for( x = 0;x<END;x++)
        {
            ARRAY[x] = '\0';
            SORTED_ARRAY[x] = '\0';
        }

        END = 0;
        ctr = 0;
        arrayLbl.setText("Array [0]:");
        statusArea.setText("Cleared!!");
    }//GEN-LAST:event_clearBttnActionPerformed

    private    void    exitBttnActionPerformed(java.awt.event.ActionEvent    evt)    {//GEN-
FIRST:event_exitBttnActionPerformed
        System.exit(0);
    }//GEN-LAST:event_exitBttnActionPerformed

    private    void    prevPosBttnActionPerformed(java.awt.event.ActionEvent    evt)    {//GEN-
FIRST:event_prevPosBttnActionPerformed
        if(prevPosBttn.isSelected())
        {
            dirComboBox.setEnabled(true);
            prevPosField.setEnabled(false);
            prevPosBttn.setText("OFF");
        }
        else
        {
            prevPosBttn.setText("ON");
```

```
            prevPosField.setEnabled(true);
            dirComboBox.setEnabled(false);
        }
    }
    private void editBttnActionPerformed(java.awt.event.ActionEvent evt) {
        try
        {
            int no = Integer.parseInt(editField.getText());
            int x = Integer.parseInt(indexField.getText());
            ARRAY[x] = no;
            statusArea.setText(statusArea.getText()+"\n"+"Edited: Array no."+x+". "+no);
            editField.setText("0");
            indexField.setText("0");
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(null,"Please     input     a     number!","Status:
Error!",JOptionPane.ERROR_MESSAGE);
        }
    }
    private void endBttnActionPerformed(java.awt.event.ActionEvent evt) {
        if(END>0)
        {
            statusArea.setText(statusArea.getText()+"\n"+"Array  input  ended  at  Array  no.
"+(END-1));
            nextBttn.setEnabled(false);
            computeBttn.setEnabled(true);
            arrayField.setEnabled(false);
            endBttn.setEnabled(false);
        }
        else
        {
            nextBttn.setForeground(defaultForeground);
            computeBttn.setForeground(defaultForeground);
            arrayField.setForeground(defaultForeground);
            endBttn.setForeground(defaultForeground);
            JOptionPane.showMessageDialog(null,"Please     input     a     number!","Status:
Error!",JOptionPane.ERROR_MESSAGE);
        }
    }

    private void nextBttnActionPerformed(java.awt.event.ActionEvent evt) {
        int SUCCESS = saveValuesToArray();
        if(SUCCESS == 0)
        {
            endBttn.setEnabled(true);
            editBttn.setEnabled(true);
            clearBttn.setEnabled(true);
        }
    }
    private void arrayFieldKeyPressed(java.awt.event.KeyEvent evt) {
        if(evt.getKeyCode() == 10)
        {
            int SUCCESS = saveValuesToArray();
            if(SUCCESS == 0)
            {
```

```
                endBttn.setEnabled(true);
                editBttn.setEnabled(true);
                clearBttn.setEnabled(true);
            }
        }
    }

    private void seekRateFieldActionPerformed(java.awt.event.ActionEvent evt) {
    }
    private void editFieldActionPerformed(java.awt.event.ActionEvent evt)
    privatevoidshowAreaBttnMouseClicked(java.awt.event.MouseEventevt){
statusArea.setText("Array List:\n");
        for(x = 0;x<END;x++)
            statusArea.setText(statusArea.getText()+"\n"+"Array no. "+x+": "+ARRAY[x]);
    }
    private void showSortedArrayBttnMouseClicked(java.awt.event.MouseEvent evt) {
        statusArea.setText("Sorted Array List:\n");
        for(x = 0;x<END;x++)
            statusArea.setText(statusArea.getText()+"\n"+"Array no"+x+": "+SORTED_ARRAY[x]);
    }

    private void clearStatusAreaBttnMouseClicked(java.awt.event.MouseEvent evt) {
        statusArea.setText("");
    }

    private void trackNoFieldKeyReleased(java.awt.event.KeyEvent evt) {
        try
        {
            int x = Integer.parseInt(trackNoField.getText());
            if(evt.getKeyCode()==10);
            else
            if(x<0)
            {
                JOptionPane.showMessageDialog(null,"Number    of    tracks    must    not    be
negative","Status:Error",JOptionPane.ERROR_MESSAGE);
                trackNoField.setText("0");
            }
            else if(END>0)
            {
                sortArray();
                if(x<SORTED_ARRAY[END-1])
                {
                    JOptionPane.showMessageDialog(null,"Track number must not be less than
the Array inputted","Status:Error",JOptionPane.ERROR_MESSAGE);
                    trackNoField.setText("0");
                }
            }
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(null,"Invalid
Input!","Status:Error",JOptionPane.ERROR_MESSAGE);
            trackNoField.setText("0");
        }
    }//GEN-LAST:event_trackNoFieldKeyReleased
```

```
    private     void     currPosFieldKeyReleased(java.awt.event.KeyEvent     evt)     {//GEN-
FIRST:event_currPosFieldKeyReleased
        try
        {
            int x = Integer.parseInt(currPosField.getText());
            int y = Integer.parseInt(trackNoField.getText());
            if(evt.getKeyCode()==10);
            else
            if((x>=y)||(x<0))
            {
                currPosField.setText("0");
                JOptionPane.showMessageDialog(null,"Input     does     not     match     the
range!","Status:Error",JOptionPane.ERROR_MESSAGE);
            }
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(null,"Invalid
Input!","Status:Error",JOptionPane.ERROR_MESSAGE);
            currPosField.setText("0");
        }
    }//GEN-LAST:event_currPosFieldKeyReleased

    private     void     prevPosFieldKeyReleased(java.awt.event.KeyEvent     evt)     {//GEN-
FIRST:event_prevPosFieldKeyReleased
                try
        {
            int x = Integer.parseInt(prevPosField.getText());
            int y = Integer.parseInt(trackNoField.getText());
            if(evt.getKeyCode()==10);
            else
            if((x>=y)||(x<0))
            {
                prevPosField.setText("0");
                JOptionPane.showMessageDialog(null,"Input     does     not     match     the
range!","Status:Error",JOptionPane.ERROR_MESSAGE);
            }
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(null,"Invalid
Input!","Status:Error",JOptionPane.ERROR_MESSAGE);
            prevPosField.setText("0");
        }
    }

    private void seekRateFieldKeyReleased(java.awt.event.KeyEvent evt) {
        try
        {
            int x = Integer.parseInt(seekRateField.getText());
            if(evt.getKeyCode()==10);
            else
            if((x<0))
            {
                seekRateField.setText("0");
```

```
                JOptionPane.showMessageDialog(null,"Must be greater than or equal to
zero!","Status:Error",JOptionPane.ERROR_MESSAGE);
            }
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(null,"Invalid
Input!","Status:Error",JOptionPane.ERROR_MESSAGE);
            seekRateField.setText("0");
        }
    }

    private void arrayFieldKeyReleased(java.awt.event.KeyEvent evt)
        try
        {
            if(evt.getKeyCode()==10);
            else
            {
                int x = Integer.parseInt(arrayField.getText());
                int y = Integer.parseInt(trackNoField.getText());
                if(((x<0)||(x>=y)))
                {
                    JOptionPane.showMessageDialog(null,"Notin
range!","Status:Error",JOptionPane.ERROR_MESSAGE);
                    arrayField.setText("0");
                }
            }
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(null,"Invalid
Input!","Status:Error",JOptionPane.ERROR_MESSAGE);
            arrayField.setText("0");
        }
    }
    public void setAccessibleContext(AccessibleContext accessibleContext) {
        this.setLocationRelativeTo(null);
    }

    public static void main(String args[]) {
        try {
            for(javax.swing.UIManager.LookAndFeelInfoinfo:
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Windows".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException | InstantiationException | IllegalAccessException |
javax.swing.UnsupportedLookAndFeelExceptionex){
java.util.logging.Logger.getLogger(launchFrame.class.getName()).log(java.util.logging.Level.
SEVERE, null, ex);
        }
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new launchFrame().setVisible(true);
```

```
        }
    });
}
private javax.swing.JLabel CurrPosLbl;
private javax.swing.JLabel PrevPosLbl;
private javax.swing.JPanel StatusPanel;
private javax.swing.JTextField arrayField;
private javax.swing.JLabel arrayLbl;
private javax.swing.ButtonGroup buttonGroup1;
private static javax.swing.JRadioButton cLookBttn;
private static javax.swing.JRadioButton cScanBttn;
private javax.swing.JButton clearBttn;
private javax.swing.JLabel clearStatusAreaBttn;
private javax.swing.JButton computeBttn;
private static javax.swing.JTextField currPosField;
private javax.swing.JPanel infoPanel;
private javax.swing.JMenu jMenu3;
private javax.swing.JPanel jPanel1;
private static javax.swing.JRadioButton lookBttn;
private javax.swing.JButton nextBttn;
private javax.swing.JPanel panel;
private javax.swing.JToggleButton prevPosBttn;
private static javax.swing.JTextField prevPosField;
private static javax.swing.JRadioButton scanBttn;
private javax.swing.JLabel scanTypeLbl;
private javax.swing.JPanel scanTypePanel;
private javax.swing.JTextField seekRateField;
private javax.swing.JLabel seekRateLbl;
private javax.swing.JLabel showAreaBttn;
private javax.swing.JLabel showSortedArrayBttn;
private static javax.swing.JRadioButton sstfBttn;
private javax.swing.JTextArea statusArea;
private static javax.swing.JTextField trackNoField;
private javax.swing.JLabel trackNoLbl;
private javax.swing.JTextField valOfAField;
private int saveValuesToArray() {
   int SUCCESS = 0;
   try
   {
       String temp_display;
       int TRACK = Integer.parseInt(trackNoField.getText());
       int NUMBER = Integer.parseInt(arrayField.getText());
       if(TRACK>0)
       temp_display  = "Number not in range! ( 0 to " +(TRACK-1)+" )";
       else
       temp_display  = "Input a number greater than 0!";
       if((NUMBER>=0)&&(NUMBER<TRACK)){
            ARRAY[ctr] = NUMBER;
            ctr++;
            END = ctr;
            temp_END = END;
            statusArea.setText(statusArea.getText()+"\n"+arrayLbl.getText()+" "+NUMBER);
            arrayLbl.setText("Array ["+ctr+"] value:");
            arrayField.setText("");

        }
```

```
            else
            {
                SUCCESS = 1;
                JOptionPane.showMessageDialog(null,""+temp_display,"Status:
Error!",JOptionPane.ERROR_MESSAGE);
            }
        }
        catch(NumberFormatException | HeadlessException e)
        {
            JOptionPane.showMessageDialog(null,"Pleaseinputanumber!","Status:
Error!",JOptionPane.ERROR_MESSAGE);
            SUCCESS = 1;
        }

        return SUCCESS;
    }

    private void fcfsProcess() {
        try
        {
            for(x = 0;x<END;x++)
                returnArray[x]=ARRAY[x];
            THM = 0;
            int TEMP = Integer.parseInt(currPosField.getText());
            int VALUE = TEMP;
            int TRACK = Integer.parseInt(trackNoField.getText());
            float SEEK_RATE = Float.parseFloat(seekRateField.getText());
            DISPLAY = "First Come First Serve\n\n";

            for( x = 0;x<END;x++)
            {
                THM+= abs(ARRAY[x]-TEMP);
                TEMP = ARRAY[x];
            }

            ST = THM * SEEK_RATE;
            DISPLAY += "Total Head Movement:" + THM + "\n\n";
            DISPLAY += "Seek Time:" + ST +"ms\n\n";
        }
        catch(Exception e)
        {
            e.printStackTrace();}
        }

    private void sortArray() {
        try
        {
            int same_number_ctr = 0;
            int currPos = Integer.parseInt(currPosField.getText());

            for(x = 0;x<END;x++)
            {
                SORTED_ARRAY[x] = ARRAY[x];
            }

            for(x = 0;x+same_number_ctr<END;x++)
```

```
        {
            if(currPos==SORTED_ARRAY[x])same_number_ctr++;
            SORTED_ARRAY[x] = SORTED_ARRAY[x+same_number_ctr];
        }
            temp_END = END;
            END-=same_number_ctr;

        for(x = 0;x<END-1;x++)
            for(int y = 0;y<END-1;y++)
                if(SORTED_ARRAY[y]>SORTED_ARRAY[y+1])
                {
                    int TEMP = SORTED_ARRAY[y];
                    SORTED_ARRAY[y] = SORTED_ARRAY[y+1];
                    SORTED_ARRAY[y+1] = TEMP;
                }
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

private void c_scanProcess() {
    try
    {
        int highestValue = Integer.parseInt(trackNoField.getText())-1;
        int prevPos = Integer.parseInt(prevPosField.getText());
        int currPos = Integer.parseInt(currPosField.getText());
        int a = Integer.parseInt(valOfAField.getText());
        float SEEK_RATE = Float.parseFloat(seekRateField.getText());
        String direction = "";
        direction += dirComboBox.getSelectedItem();
        String a_string = "";
        DISPLAY = "C-SCAN\n\n";

        for(x = 0;x<END;x++)
            if((SORTED_ARRAY[x]<currPos)&&(currPos<=SORTED_ARRAY[x+1]))
                    break;

        if(prevPosField.isEnabled() == true)
        {
            if(prevPos<currPos)
            {
                THM = abs(highestValue-currPos) + abs(SORTED_ARRAY[x]);
                DIRECTION = 1;
            }
            else
                THM = abs(currPos) + abs(SORTED_ARRAY[x+1]-highestValue);
        }
        else
        {
            if(direction.equals("UPWARD"))
            {
                THM = abs(currPos-highestValue) + abs(SORTED_ARRAY[x]);
                DIRECTION = 1;
            }
```

```
            else
                THM = abs(currPos) + abs(SORTED_ARRAY[x+1]-highestValue);
        }

        if( DIRECTION == 1 )
        {
            DIRECTION = 0;
            returnArray[0] = highestValue;
            returnArray[1] = 0;
            returnArray[2] = SORTED_ARRAY[x];
        }
        else
        {
            returnArray[0] = 0;
            returnArray[1] = highestValue;
            returnArray[2] = SORTED_ARRAY[x+1];
        }

        if(a != 0)
            THM += a;
        else
            a_string = " + a";


        ST = (THM-a) * SEEK_RATE;
        DISPLAY += "Total Head Movement: " + THM + a_string + "\n\n";
        DISPLAY += "Seek Time:" + ST +"ms\n\n";
    }
    catch(Exception e)
    {
       e.printStackTrace();
    }
}

private void scanProcess() {
    try
    {
        int highestValue = Integer.parseInt(trackNoField.getText())-1;
        int prevPos = Integer.parseInt(prevPosField.getText());
        int currPos = Integer.parseInt(currPosField.getText());
        int SEEK_RATE = Integer.parseInt(seekRateField.getText());
        int lowestValue = 0;
        String direction = "";
        direction += dirComboBox.getSelectedItem();
        String a_string = "";
        DISPLAY = "SCAN\n\n";

        if(prevPosField.isEnabled() == true)
        {
            if(prevPos<currPos)
            {
                THM = abs(currPos-highestValue) + abs(highestValue-SORTED_ARRAY[0]);
                DIRECTION = 1;
            }
            else
            {
```

```
                THM = abs(currPos-0) + abs(SORTED_ARRAY[END-1]-0);
            }

        }
        else
        {
            if(direction.equals("UPWARD"))
            {
                THM = abs(currPos-highestValue) + abs(highestValue-SORTED_ARRAY[0]);
                DIRECTION = 1;
            }
            else
                THM = abs(currPos-0) + abs(SORTED_ARRAY[END-1]-0);
        }

        if( DIRECTION == 1 )
        {
            returnArray[0]= highestValue;
            returnArray[1]= SORTED_ARRAY[0];
            DIRECTION = 0;
        }
        else
        {
            returnArray[0]= 0;
            returnArray[1]= SORTED_ARRAY[END-1];
        }


        ST = THM * SEEK_RATE;
        DISPLAY += "Total Head Movement: " + THM + "\n\n";
        DISPLAY += "Seek Time:" + ST +"ms\n\n";
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

private void lookProcess() {
    try
    {
        int prevPos = Integer.parseInt(prevPosField.getText());
        int currPos = Integer.parseInt(currPosField.getText());
        float SEEK_RATE = Float.parseFloat(seekRateField.getText());
        int lowestValue = 0;
        String direction = "";
        direction += dirComboBox.getSelectedItem();
        String a_string = "";
        DISPLAY = "LOOK\n\n";

        returnArray[0]=ARRAY[0];
        returnArray[1]=ARRAY[END-1];

        if(prevPosField.isEnabled() == true)
        {
            if(prevPos<currPos)
```

```
                    TH=abs(currPos-SORTED_ARRAY[END-1])+abs(SORTED_ARRAY[END-1]-
SORTED_ARRAY[0]);
                else
                {
                    THM=abs(currPos-SORTED_ARRAY[0])+abs(SORTED_ARRAY[END-1]-
SORTED_ARRAY[0]);
                    DIRECTION = 1;
                }
            }
            else
            {
                if(direction.equals("UPWARD"))
                {
                    THM=abs(currPos-SORTED_ARRAY[END-1])+abs(SORTED_ARRAY[END-1]-
SORTED_ARRAY[0]);
                    DIRECTION = 1;
                }
                else
                    THM=abs(currPos-SORTED_ARRAY[0])+abs(SORTED_ARRAY[END-1]-
SORTED_ARRAY[0]);
            }
            if( DIRECTION == 1 )
            {
                DIRECTION = 0;
                returnArray[0]= SORTED_ARRAY[0];
                returnArray[1]= SORTED_ARRAY[END-1];
            }
            else
            {
                returnArray[0]= SORTED_ARRAY[END-1];
                returnArray[1]= SORTED_ARRAY[0];
            }

            ST = THM * SEEK_RATE;
            DISPLAY += "Total Head Movement: " + THM + "\n\n";
            DISPLAY += "Seek Time:" + ST +"ms\n\n";
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    private void c_lookProcess() {
         try
         {
            int prevPos = Integer.parseInt(prevPosField.getText());
            int currPos = Integer.parseInt(currPosField.getText());
            int a = Integer.parseInt(valOfAField.getText());
            float SEEK_RATE = Float.parseFloat(seekRateField.getText());
            String direction = "";
            direction += dirComboBox.getSelectedItem();
            String a_string = "";
            DISPLAY = "C-LOOK\n\n";
            for(x = 0;x<END;x++)
                if((SORTED_ARRAY[x]<currPos)&&(currPos<SORTED_ARRAY[x+1]))
```

```
                    break;

            if(prevPosField.isEnabled() == true)
            {
                if(prevPos<currPos)
                {
                    THM=abs(currPos-SORTED_ARRAY[END-1])+abs(SORTED_ARRAY[0]-
SORTED_ARRAY[x]);
                   DIRECTION = 1;
                }
                else
                    THM=abs(currPos-SORTED_ARRAY[0])+abs(SORTED_ARRAY[x+1]-SORTED_ARRAY[END-
1]);
            }
            else
            {
                if(direction.equals("UPWARD"))
                {
                    THM=abs(currPos-SORTED_ARRAY[END-1])+abs(SORTED_ARRAY[0]-
SORTED_ARRAY[x]);
                    DIRECTION = 1;
                }
                else
                    THM=abs(currPos-SORTED_ARRAY[0])+abs(SORTED_ARRAY[x+1]-SORTED_ARRAY[END-
1]);
            }

            if(a != 0)
                THM += a;
            else
                a_string = " + a";

            if( DIRECTION == 1 )
            {
                DIRECTION = 0;
                returnArray[0] = SORTED_ARRAY[END-1];
                returnArray[1] = SORTED_ARRAY[0];
                returnArray[2] = SORTED_ARRAY[x];
            }
            else
            {
                returnArray[0] = SORTED_ARRAY[0];
                returnArray[1] = SORTED_ARRAY[END-1];
                returnArray[2] = SORTED_ARRAY[x+1];
            }

            ST = (THM-a) * SEEK_RATE;
            DISPLAY += "Total Head Movement: " + THM + a_string + "\n\n";
            DISPLAY += "Seek Time:" + ST +"ms\n\n";
         }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
```

```
private void sstfProcess() {
    try
    {
        int prevPos = Integer.parseInt(prevPosField.getText());
        int currPos = Integer.parseInt(currPosField.getText());
        float SEEK_RATE = Float.parseFloat(seekRateField.getText());
        int upward_direction = 0;
        DISPLAY = "Shortest Seek Time First\n\n";
        int ctr = 0;
        if(prevPos<currPos)
            upward_direction= 1;
        int LOWER = 0;
        int HIGHER = 0;
        int MID = 0;
        int repeat = 0;
        int temp_MID = x;

        for(x = 1;x<END;x++)
        if((SORTED_ARRAY[x-1]<=currPos)&&(currPos<SORTED_ARRAY[x]))
        {
            if(abs(currPos-SORTED_ARRAY[x-1])==abs(currPos-SORTED_ARRAY[x]))
            {
                if(upward_direction==0)
                {
                    LOWER = x-2;
                    MID = x-1;
                    HIGHER = x;
                    returnArray[ctr++] = SORTED_ARRAY[x-1];
                }
                else
                {
                    LOWER = x-1;
                    MID = x;
                    HIGHER = x+1;
                    returnArray[ctr++] = SORTED_ARRAY[x];
                }
            }
            else if(abs(currPos-SORTED_ARRAY[x-1])<abs(currPos-SORTED_ARRAY[x]))
            {
                LOWER = x-2;
                MID = x-1;
                HIGHER = x;
                returnArray[ctr++] = SORTED_ARRAY[x-1];
            }
            else
            {
                LOWER = x-1;
                MID = x;
                HIGHER = x+1;
                returnArray[ctr++] = SORTED_ARRAY[x];
            }
            break;
        }

            temp_MID = MID;
```

```
            while(ctr<END)
            {
                if((LOWER<0)||(HIGHER==END))
                {
                    if((LOWER<0)&&(HIGHER<END))
                    {
                        while(ctr<END)
                        {
                            returnArray[ctr++] = SORTED_ARRAY[HIGHER++];
                            if((repeat == 0)&&(HIGHER>=END))
                            {
                                returnArray[ctr++] = SORTED_ARRAY[END-1];
                                repeat = 1;
                            }
                        }
                    }
                    else if((HIGHER==END)&&(LOWER>=0))
                        while(ctr<END)
                        {
                            returnArray[ctr++] = SORTED_ARRAY[LOWER--];
                        }
                }
                else
                {
                   if(abs(SORTED_ARRAY[MID]-SORTED_ARRAY[LOWER])<abs(SORTED_ARRAY[MID]-
SORTED_ARRAY[HIGHER]))
                   {
                       if((temp_MID==x)||(temp_MID == x-1))
                            returnArray[ctr++] = SORTED_ARRAY[LOWER];
                       else
                           returnArray[ctr++] = SORTED_ARRAY[MID];
                        MID = LOWER--;
                   }
                   else  if(abs(SORTED_ARRAY[MID]-SORTED_ARRAY[LOWER])>abs(SORTED_ARRAY[MID]-
SORTED_ARRAY[HIGHER]))
                   {
                        if((temp_MID==x)||(temp_MID == x-1))
                        {
                            returnArray[ctr++] = SORTED_ARRAY[HIGHER];
                            MID = HIGHER++;
                        }
                        else
                        {
                            returnArray[ctr++] = SORTED_ARRAY[MID];
                             MID = HIGHER++;

                            if((repeat == 0)&&(HIGHER>=END))
                            {
                                returnArray[ctr++] = SORTED_ARRAY[END-1];
                                repeat = 1;
                            }
                        }
                   }
                   elseif(abs(SORTED_ARRAY[MID]-SORTED_ARRAY[LOWER])==abs(SORTED_ARRAY[MID]-
SORTED_ARRAY[HIGHER]))
                   {
```

```
                    if(upward_direction == 0)
                     {
                         if((temp_MID==x)||(temp_MID == x-1))
                              returnArray[ctr++] = SORTED_ARRAY[LOWER];
                         else
                             returnArray[ctr++] = SORTED_ARRAY[MID];
                          MID = LOWER--;
                     }
                    else
                    {
                         if((temp_MID==x)||(temp_MID == x-1))
                         {
                             returnArray[ctr++] = SORTED_ARRAY[HIGHER];
                             MID = HIGHER++;
                         }
                         else
                         {
                             returnArray[ctr++] = SORTED_ARRAY[MID];
                              MID = HIGHER++;

                             if((repeat == 0)&&(HIGHER>=END))
                             {
                                 returnArray[ctr++] = SORTED_ARRAY[END-1];
                                 repeat = 1;
                             }
                         }
                    }
                }
            }
        }

        THM=abs(currPos-returnArray[0]);
        for(ctr=1;ctr<END;ctr++)
            THM+=abs(returnArray[ctr-1]-returnArray[ctr]);


        ST = THM * SEEK_RATE;
        DISPLAY += "Total Head Movement: " + THM + "\n\n";
        DISPLAY += "Seek Time:" + ST +"ms\n\n";
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

public static int returnArray(int a)
{
    return(returnArray[a]);
}

public static int returnLimit()
{
    int LIMIT = Integer.parseInt(trackNoField.getText());
    return LIMIT;
}
```

```
    public static String returnDisplay()
    {
        return DISPLAY;
    }

    public static int returnArrayCount()
    {
        return END;
    }

    public static int returnCurrentPosition()
    {
        final  int currPos = Integer.parseInt(currPosField.getText());
        return currPos;
    }

    public static int returnPreviousPosition()
    {
        final int prevPos = Integer.parseInt(prevPosField.getText());
        return prevPos;
    }

    public static int returnChosenProcess()
    {
        if(fcfsBttn.isSelected()==true)
            return 1;
        else
        if(sstfBttn.isSelected()==true)
            return 2;
        else
        if(scanBttn.isSelected()==true)
            return 3;
        else
        if(lookBttn.isSelected()==true)
            return 4;
        else
            return 5;
    }
}
```

### 9.1.2. LogCreator.java

```
package DiskScheduling;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
public class LogCreator extends javax.swing.JFrame {
    public LogCreator() {
        initComponents();
        setVisible(true);
    }
    private void initComponents() {
```

```
        saveBttn = new javax.swing.JButton();
        discardBttn = new javax.swing.JButton();
        jButton3 = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        logArea = new javax.swing.JTextArea();
        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Log Editor");
        saveBttn.setText("Save");
        saveBttn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                saveBttnActionPerformed(evt);
            }
        });

        discardBttn.setText("Discard");
        discardBttn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                discardBttnActionPerformed(evt);
            }
        });
        jButton3.setText("Find & Update");
        jButton3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton3ActionPerformed(evt);
            }
        });
        logArea.setColumns(20);
        logArea.setRows(5);
        jScrollPane1.setViewportView(logArea);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
             .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(saveBttn, javax.swing.GroupLayout.PREFERRED_SIZE, 136,
javax.swing.GroupLayout.PREFERRED_SIZE)        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jScrollPane1,    javax.swing.GroupLayout.PREFERRED_SIZE,    335,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE, false)
                    .addComponent(discardBttn,          javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jButton3,             javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(saveBttn,             javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap())
```

```
        );

        pack();
    }
    private void saveBttnActionPerformed(java.awt.event.ActionEvent evt) {
        String log_txt = logArea.getText();
        try
        {
            BufferedWriter bw = new BufferedWriter(new FileWriter(new File("log.txt")));
            bw.append(log_txt);
            bw.close();
            JOptionPane.showMessageDialog(null,"log
updated","Status!",JOptionPane.INFORMATION_MESSAGE);
            dispose();
        }
        catch(Exception e)
        {
            File f = new File("log.txt");
            f.mkdir();
        }
    }
    private void discardBttnActionPerformed(java.awt.event.ActionEvent evt) {
        dispose();}
    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt){
        JFileChooser file = new JFileChooser();
        file.setAcceptAllFileFilterUsed(true);
        int result = file.showOpenDialog(null);
        if (result == JFileChooser.APPROVE_OPTION)
        {
                File selectedFile = file.getSelectedFile();
                String filename = selectedFile.getAbsolutePath();
                String line = "";
                try {
                     BufferedReader br = new BufferedReader(new FileReader(filename));
                        while(( line = br.readLine())!= null)
                             logArea.setText(logArea.getText()+"\n"+line);
                        } catch (Exception ex) {
                            Logger.getLogger(LogCreator.class.getName()).log(Level.SEVERE,
null, ex);
                        }
        }
        if(result == JFileChooser.CANCEL_OPTION)
        {
                file.setVisible(false);
        }
    }
    public static void main(String args[]) {
        try
            for(javax.swing.UIManager.LookAndFeelInfoinfo:
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName()))
 {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
```

```
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(LogCreator.class.getName()).log(java.util.logging.Level.S
EVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(LogCreator.class.getName()).log(java.util.logging.Level.S
EVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(LogCreator.class.getName()).log(java.util.logging.Level.S
EVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(LogCreator.class.getName()).log(java.util.logging.Level.S
EVERE, null, ex);
        }

        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new LogCreator().setVisible(true);
            }
        });
    }
    private javax.swing.JButton discardBttn;
    private javax.swing.JButton jButton3;
    private javax.swing.JScrollPane jScrollPane1;
    public javax.swing.JTextArea logArea;
    private javax.swing.JButton saveBttn;
}
```

### 9.1.3. ResultingWindows.java

```
package DiskScheduling;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.Graphics;
public class ResultingWindow extends javax.swing.JFrame {
    int[] nmbr = new int[1000];
    int LIMIT;
    int SCALE = 300;
    int END;
    String DISPLAY = " ";
    int prevPos;
    int prevPosScale;
    int currPos;
    int currPosScale;
    public ResultingWindow() {
        getNumbers();
        setDisplay();
        getPrevPos();
        getCurrPos();
        setVisible(true);
        setSize(693, 386);
        initComponents();
```

```
    }
        private void initComponents() {
        ResultingPanel = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        resultArea = new javax.swing.JTextArea();
        jLabel1 = new javax.swing.JLabel();
        jPanel1 = new javax.swing.JPanel();
        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Result");
        setResizable(false);
        ResultingPanel.setBackground(new java.awt.Color(0, 51, 51));
        ResultingPanel.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        resultArea.setEditable(false);
        resultArea.setBackground(new java.awt.Color(0, 51, 51));
        resultArea.setColumns(20);
        resultArea.setFont(new java.awt.Font("Times New Roman", 0, 12)); // NOI18N
        resultArea.setForeground(new java.awt.Color(255, 255, 255));
        resultArea.setRows(5);
resultArea.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactory.
createEtchedBorder(javax.swing.border.EtchedBorder.RAISED,              java.awt.Color.white,
java.awt.Color.white),"Result",javax.swing.border.TitledBorder.CENTER,
javax.swing.border.TitledBorder.TOP,newjava.awt.Font("Tahoma",0,11),  new  java.awt.Color(255,
255, 255)));
        jScrollPane1.setViewportView(resultArea);
        jLabel1.setFont(new java.awt.Font("Times New Roman", 1, 24)); // NOI18N
        jLabel1.setForeground(new java.awt.Color55, 255, 255));
        jLabel1.setText("GRAPH");
        jPanel1.setBackground(new java.awt.Color(0, 51, 51));
        jPanel1.setBorder(javax.swing.BorderFactory.createEtchedBorder(java.awt.Color.white,
java.awt.Color.white));
        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 429, Short.MAX_VALUE)
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 344, Short.MAX_VALUE)
        );
        javax.swing.GroupLayoutResultingPanelLayout=new
javax.swing.GroupLayout(ResultingPanel);
        ResultingPanel.setLayout(ResultingPanelLayout);
        ResultingPanelLayout.setHorizontalGroup(
         ResultingPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
ResultingPanelLayout.createSequentialGroup()
                .addContainerGap(18,Short.MAX_VALUE)
.addGroup(ResultingPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
              .addComponent(jPanel1,javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE,javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
ResultingPanelLayout.createSequentialGroup()
```

```
                        .addComponent(jLabel1,javax.swing.GroupLayout.PREFERRED_SIZE,    115,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(176, 176, 176)))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(jScrollPane1,javax.swing.GroupLayout.PREFERRED_SIZE,       210,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18))
        ); ResultingPanelLayout.setVerticalGroup(

ResultingPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
ResultingPanelLayout.createSequentialGroup()
                .addComponent(jLabel1,javax.swing.GroupLayout.PREFERRED_SIZE,             30,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(ResultingPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
                    .addComponent(jScrollPane1)
                    .addComponent(jPanel1,javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap())
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(ResultingPanel,javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(ResultingPanel,javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        );

        pack();
    }
    public void paint(Graphics g)
    {
       super.paint(g);
        g.setColor(Color.WHITE);
        int startx = 100, starty = 90,ctr = 0;
       g.drawString(Integer.toString(0), startx, starty);
       g.drawString(Integer.toString(LIMIT), startx+SCALE, starty);
       g.drawLine( startx , starty, startx+SCALE, starty );
       starty+=40;
        g.drawString("Prev-"+Integer.toString(launchFrame.returnPreviousPosition()),
setScale(prevPos)+startx, starty);
        g.drawLine(  setScale(prevPos)+startx  ,  starty,  setScale(currPos)+startx,  starty+10
);
        starty+=10;
        g.drawString("Cur-"+Integer.toString(launchFrame.returnCurrentPosition()),
setScale(currPos)+startx+5, starty);
```

```
        g.drawLine( setScale(currPos)+startx , starty, setScale(nmbr[ctr])+startx, starty+10
);
        starty+=10;
        for(ctr = 0; ctr<END-1; ctr++)
        {
          g.drawString(Integer.toString(nmbr[ctr]), setScale(nmbr[ctr])+startx+5, starty);
          g.drawLine(  setScale(nmbr[ctr])+startx   ,  starty,  setScale(nmbr[ctr+1])+startx,
starty+10 );
          starty+=10;
        }
        if(launchFrame.returnChosenProcess()==5)
        g.drawString("a", ((setScale(nmbr[1])+setScale(nmbr[2]))/2)+startx, starty-10);
        g.drawString(Integer.toString(nmbr[ctr]), setScale(nmbr[ctr])+startx, starty);
        resultArea.setText(DISPLAY);
    }
    public final int setScale(int nmbr)
    {
        float temp = nmbr;
        temp/=LIMIT;
        temp*=SCALE;
        return Math.round(temp);
    }


    public javax.swing.JPanel ResultingPanel;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JScrollPane jScrollPane1;
    public javax.swing.JTextArea resultArea;
    private void getNumbers(){
        if((launchFrame.returnChosenProcess()==3)||(launchFrame.returnChosenProcess()== 4))
            END = 2;
        else if(launchFrame.returnChosenProcess() == 5)
            END = 3;
        else END = launchFrame.returnArrayCount();
        for(int ctr = 0; ctr<END;ctr++)
        {
            nmbr[ctr] = launchFrame.returnArray(ctr);

        }
        LIMIT = launchFrame.returnLimit();
    }

    public void setDisplay(){
        DISPLAY = launchFrame.returnDisplay();
    }

    public void getPrevPos(){
        prevPos = launchFrame.returnPreviousPosition();
    }

    public void getCurrPos() {
        currPos = launchFrame.returnCurrentPosition();
    }
}
```

# CHAPTER-10
# SCREENSHOTS

## 10.1. OUTPUT SCREENSHOTS:

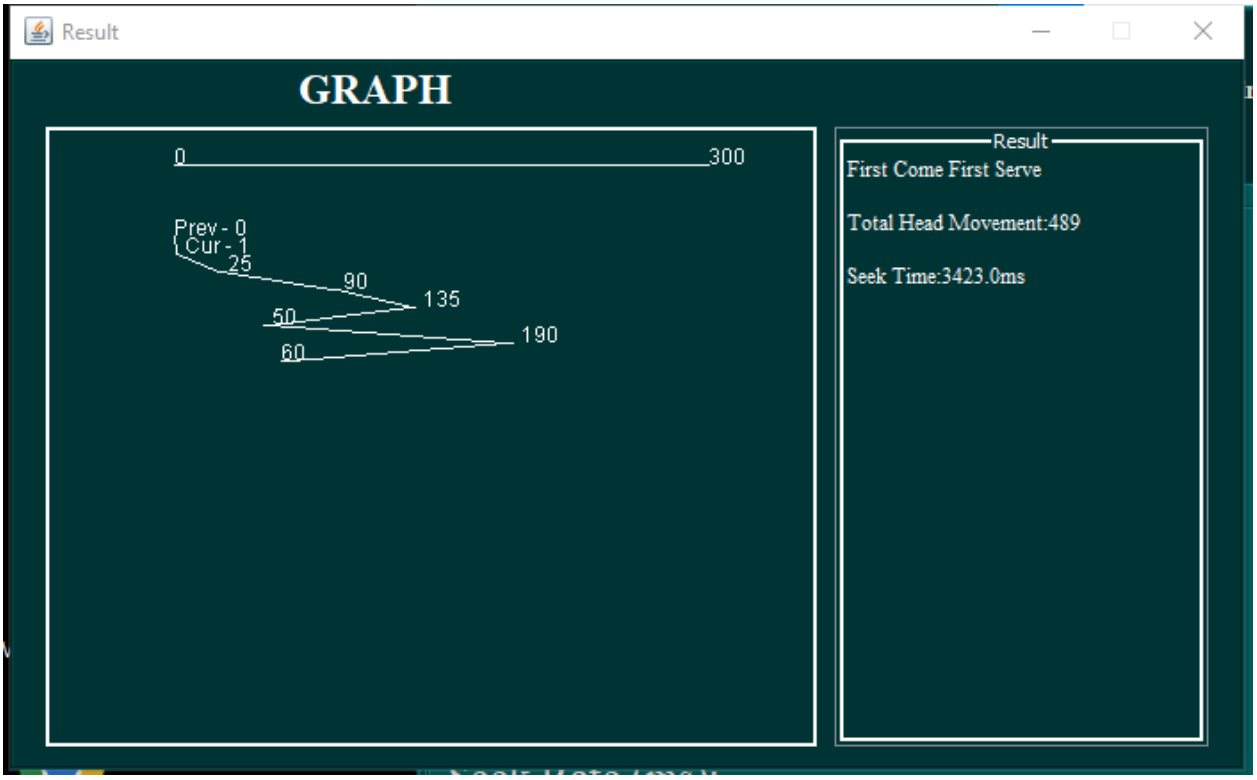


Figure 10.1.1:Result Of FCFS

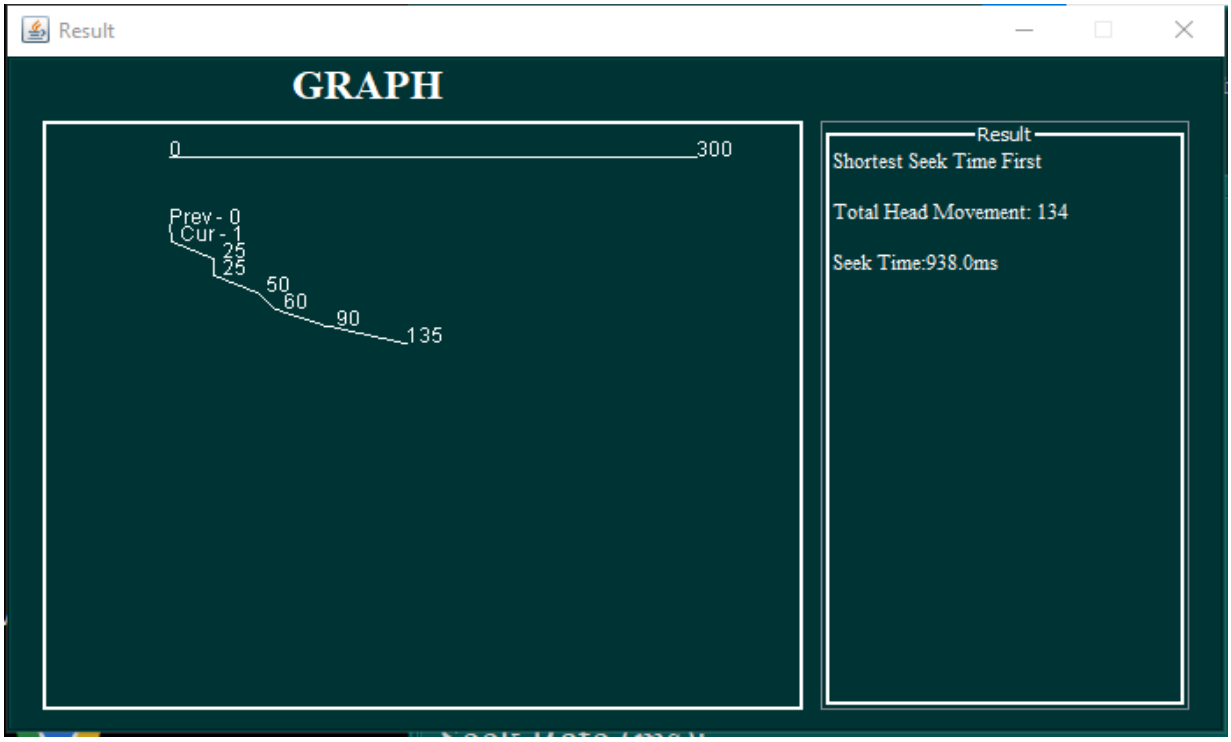


Figure 10.1.2: Result Of Shortest Seek Time First

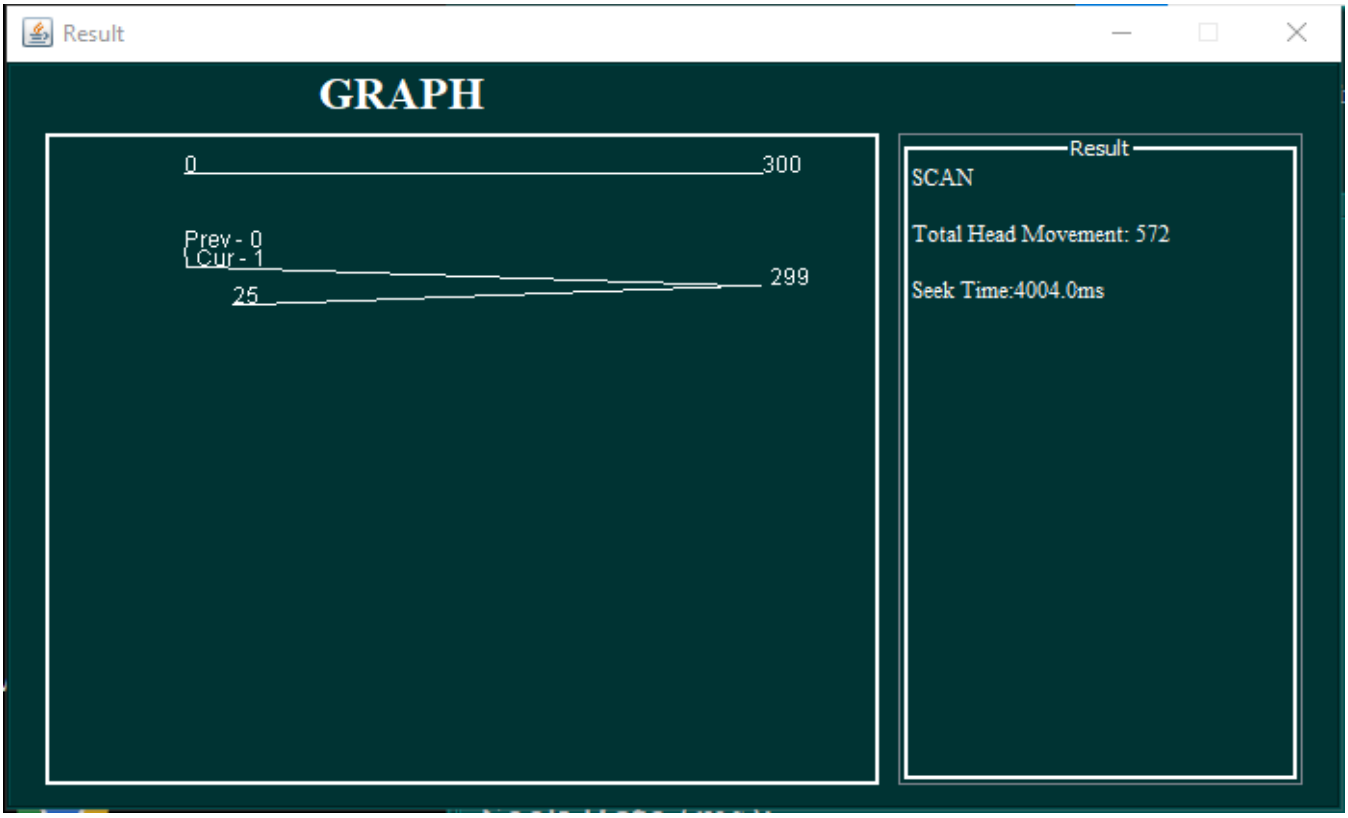


Figure 10.1.3: Result Of Scan Scheduling

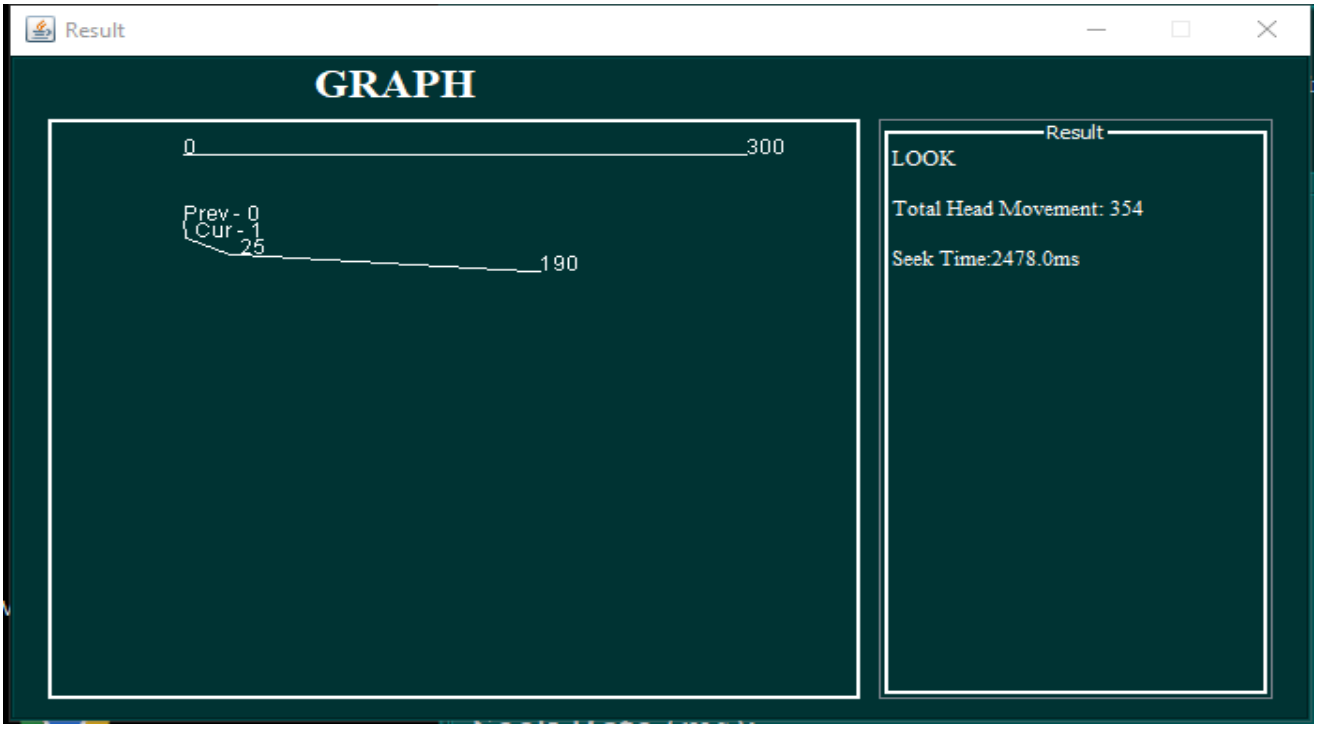


Figure 10.1.4: Look Scheduling

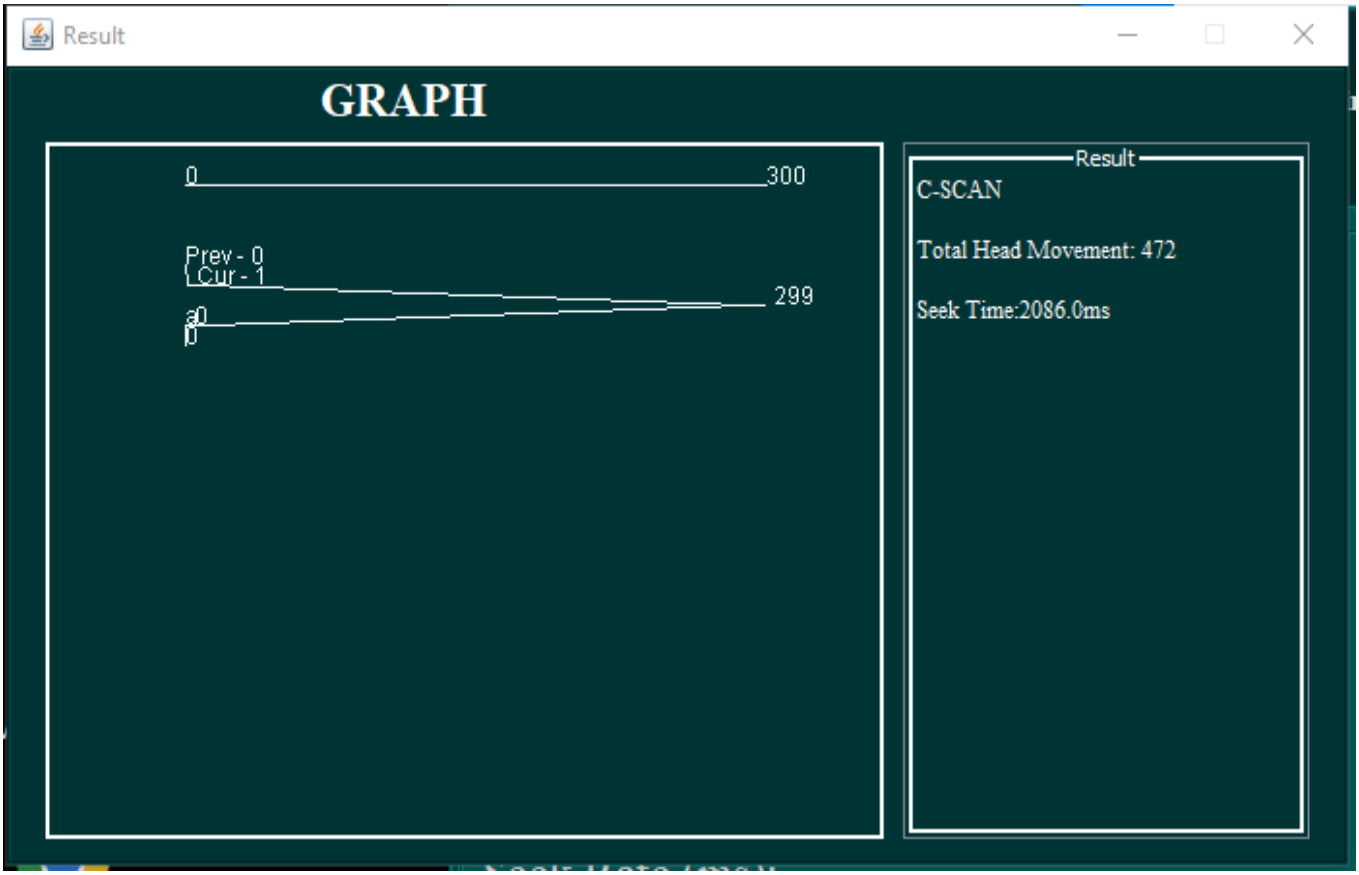


Figure 10.1.5: Result Of C-Scan Scheduling

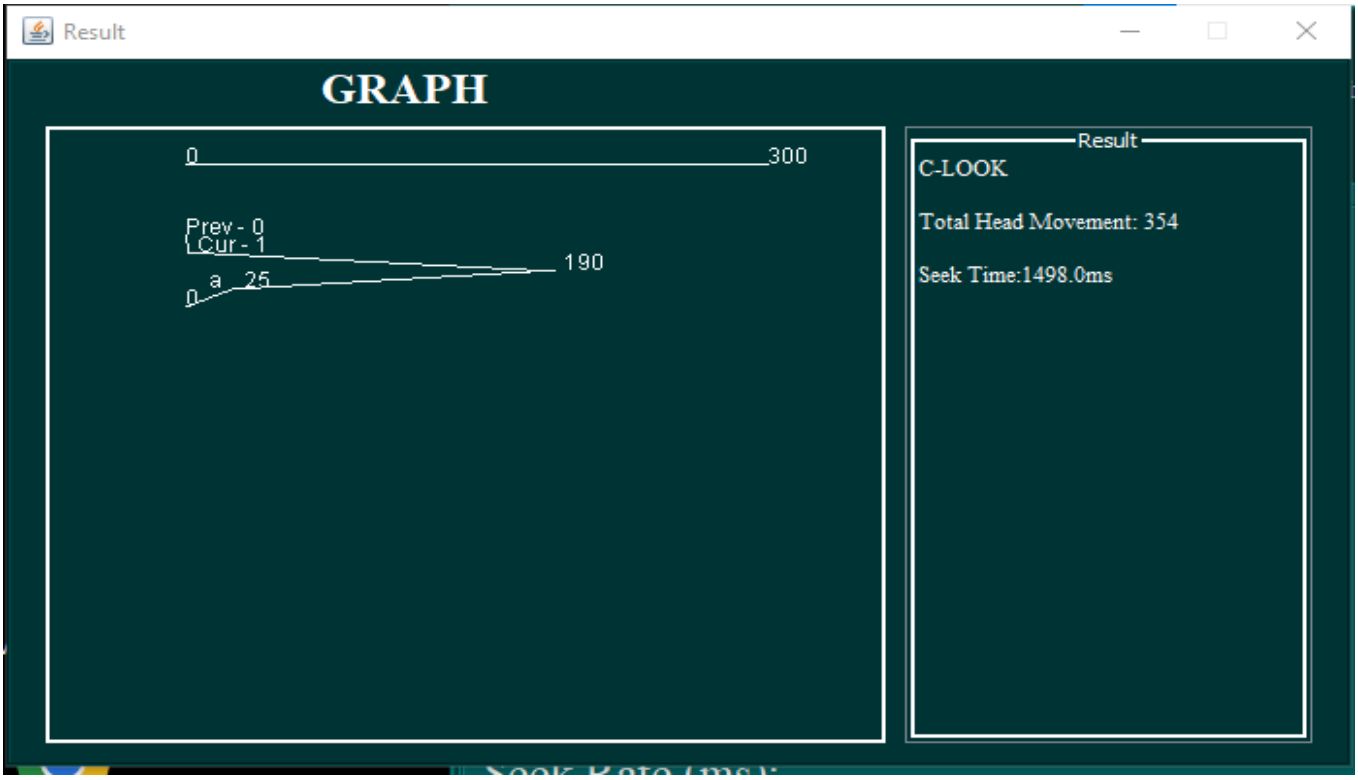


Figure 10.1.6: Result Of C-Look Scheduling

# CHAPTER-11
# Conclusion

## 11.1. Conclusion:

When selecting a Disk Scheduling algorithm, performance depends heavily on the total number of head movement and seek time. The algorithm which gives minimum seek time is better algorithm. Analysis of different disk scheduling algorithms has clearly shown that in the performance. We have calculated the average total head movement after entering the various runs for the 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 i/n context to total head movement.

FCFS is having worst performance as compared to others by producing the maximum number of cylinder jumps. And as we know that Operating System generally uses same algorithm for servicing the all set of requests irrespective of number of requests in the request set.So, from the comparison of disk scheduling algorithms in different cases, it is found that C-LOOK is the most efficient algorithm compared to FCFS, SSTF, SCAN, C- SCAN and LOOK. The total head movement and average seek time has been improvised by the C-LOOK algorithm which increases the efficiency of the disk performance.

### 11.1.1. Limitation:

When input blocks are stay in ascending order without sorting and head stay in starting block then FCFS is best.

It is not applicable for priority based requests. All requests are independent of each other and have equal priority.

This study is based on limited number of test cases.

Reliability of the research is completely based on the average seek time and total disk head movement only.

The conclusion drawn from this study may not apply in all the aspects and circumstances.

# CHAPTER-12
# REFERENCES

## 12.1. BOOKS REFERRED:

The following books were used extensively for the project development andimplementation.


- Abraham Silberschatz, Peter B. Galvin and Greg Gagne, Operating Syst em Concepts, Eight ed., UK: Wiley, 2010.
- John Ryan Celis et. al., "A Comprehensive Review for Disk Scheduling Algorithms," International Journal of Computer Science, vol. 11, issue 1, no. 1, pp. 74-79, Jan 2014.
- C. Mallikarjuna and P. C. Babu, "Performance Analysis of Disk Scheduling Algorithms," no. 05, pp. 180–184, 2016.
- J. R. Celis, D. Gonzales, E. Lagda, and L. R. Jr, "A Comprehensive Review for Disk Scheduling Algorithms," vol. 11, no. 1, pp. 74–79, 2014.
- D. Singh, "A New Optimized Real-Time Disk Scheduling Algorithm," vol. 93, no. 18, pp. 7–12, 2014.
- Dr. Muhammad Younus Javed and Mr. Ihsan Ullah Khan, "Simulation and Performance Comparison of Four Disk Scheduling Algorithms", IEEE, 0-7803-6355-8/1 0.00 02000.
- Peter Baer Galvin, Abraham Silverschatz, Greg Gagne,"Operating System Concepts", John Willey & Sons, Inc., Sixth Edition, 2002.


## 12.2. WEBSITES REFERRED

The following links were searched and exploited extensively for the project development and implementation.


- https://github.com/gjhuerte/disk-scheduling.git
- https://workat.tech/core-cs/tutorial/disk-scheduling-algorithms-in-operating-system-os-ope5ahnn6mhh
- https://www.javatpoint.com/os-disk-scheduling