**Name : Tanishq Thuse**

**Branch : CS(AI)**

**Year : SY**

**Div : B**

**Roll no. : 60**

**PRN : 12310237**

**Title : OS Assignment-8  I/O and File Management**

**Q1) FCFS**

**Code :**

*// Java program to demonstrate*

*// FCFS Disk Scheduling algorithm*

*class* fcfs

{

*static* int size = 8;

*static* void FCFS(int arr[], int head)

{

    int seek\_count = 0;

    int distance, cur\_track;

    for (int i = 0; i < size; i++)

    {

        cur\_track = arr[i];

*// calculate absolute distance*

        distance = Math.abs(cur\_track - head);

*// increase the total count*

        seek\_count += distance;

*// accessed track is now new head*

        head = cur\_track;

    }

    System.out.println("Total number of " +

                    "seek operations = " +

                        seek\_count);

*// Seek sequence would be the same*

*// as request array sequence*

    System.out.println("Seek Sequence is");

    for (int i = 0; i < size; i++)

    {

        System.out.println(arr[i]);

    }

}

*// Driver code*

*public* *static* void main(String[] args)

{

*// request array*

    int arr[] = { 176, 79, 34, 60,

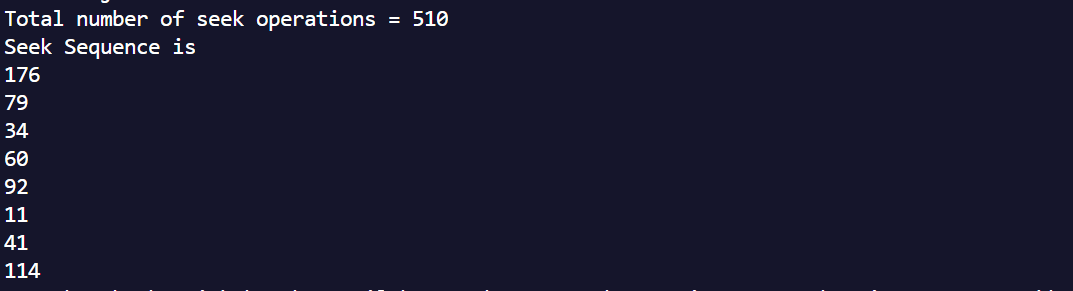
                92, 11, 41, 114 };

    int head = 50;

    FCFS(arr, head);

}

**Output :**



**Q2 ) SSTF**

**Code :**

// Java program for implementation of

// SSTF disk scheduling

public class sstf {

    static class node {

        // represent difference between

        // head position and track number

        int distance = 0;

        // true if track has been accessed

        boolean accessed = false;

    }

    // Calculates difference of each

    // track number with the head position

    public static void calculateDifference(int queue[],

                                        int head, node diff[])

    {

        for (int i = 0; i < diff.length; i++)

            diff[i].distance = Math.abs(queue[i] - head);

    }

    // find unaccessed track

    // which is at minimum distance from head

    public static int findMin(node diff[])

    {

        int index = -1, minimum = Integer.MAX\_VALUE;

        for (int i = 0; i < diff.length; i++) {

            if (!diff[i].accessed && minimum > diff[i].distance) {

                minimum = diff[i].distance;

                index = i;

            }

        }

        return index;

    }

    public static void shortestSeekTimeFirst(int request[],int head)

    {

        if (request.length == 0)

            return;

        // create array of objects of class node

        node diff[] = new node[request.length];

        // initialize array

        for (int i = 0; i < diff.length; i++)

            diff[i] = new node();

        // count total number of seek operation

        int seek\_count = 0;

        // stores sequence in which disk access is done

        int[] seek\_sequence = new int[request.length + 1];

        for (int i = 0; i < request.length; i++) {

            seek\_sequence[i] = head;

            calculateDifference(request, head, diff);

            int index = findMin(diff);

            diff[index].accessed = true;

            // increase the total count

            seek\_count += diff[index].distance;

            // accessed track is now new head

            head = request[index];

        }

        // for last accessed track

        seek\_sequence[seek\_sequence.length - 1] = head;

        System.out.println("Total number of seek operations = "

                                                    + seek\_count);

        System.out.println("Seek Sequence is");

        // print the sequence

        for (int i = 0; i < seek\_sequence.length; i++)

            System.out.println(seek\_sequence[i]);

    }

    public static void main(String[] args)

    {

        // request array

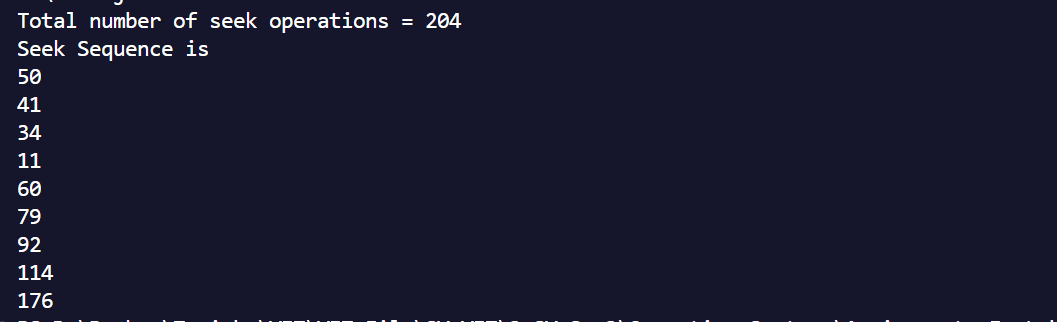
        int arr[] = { 176, 79, 34, 60, 92, 11, 41, 114 };

        shortestSeekTimeFirst(arr, 50);

    }

}

**Output :**



**Q3) SCAN**

**Code :**

import *java.util.\**;

*class* scan

{

*static* int size = 8;

*static* int disk\_size = 200;

*static* void SCAN(int arr[], int head, String direction)

{

    int seek\_count = 0;

    int distance, cur\_track;

    Vector<Integer> left = new Vector<Integer>(),

                    right = new Vector<Integer>();

    Vector<Integer> seek\_sequence = new Vector<Integer>();

*// appending end values*

*// which has to be visited*

*// before reversing the direction*

    if (direction == "left")

        left.add(0);

    else if (direction == "right")

        right.add(disk\_size - 1);

    for (int i = 0; i < size; i++)

    {

        if (arr[i] < head)

            left.add(arr[i]);

        if (arr[i] > head)

            right.add(arr[i]);

    }

*// sorting left and right vectors*

    Collections.sort(left);

    Collections.sort(right);

*// run the while loop two times.*

*// one by one scanning right*

*// and left of the head*

    int run = 2;

    while (run-- >0)

    {

        if (direction == "left")

        {

            for (int i = left.size() - 1; i >= 0; i--)

            {

                cur\_track = left.get(i);

*// appending current track to seek sequence*

                seek\_sequence.add(cur\_track);

*// calculate absolute distance*

                distance = Math.abs(cur\_track - head);

*// increase the total count*

                seek\_count += distance;

*// accessed track is now the new head*

                head = cur\_track;

            }

            direction = "right";

        }

        else if (direction == "right")

        {

            for (int i = 0; i < right.size(); i++)

            {

                cur\_track = right.get(i);

*// appending current track to seek sequence*

                seek\_sequence.add(cur\_track);

*// calculate absolute distance*

                distance = Math.abs(cur\_track - head);

*// increase the total count*

                seek\_count += distance;

*// accessed track is now new head*

                head = cur\_track;

            }

            direction = "left";

        }

    }

    System.out.print("Total number of seek operations = "

                        + seek\_count + "\n");

    System.out.print("Seek Sequence is" + "\n");

    for (int i = 0; i < seek\_sequence.size(); i++)

    {

        System.out.print(seek\_sequence.get(i) + "\n");

    }

}

*// Driver code*

*public* *static* void main(String[] args)

{

*// request array*

    int arr[] = { 176, 79, 34, 60,

                    92, 11, 41, 114 };

    int head = 50;

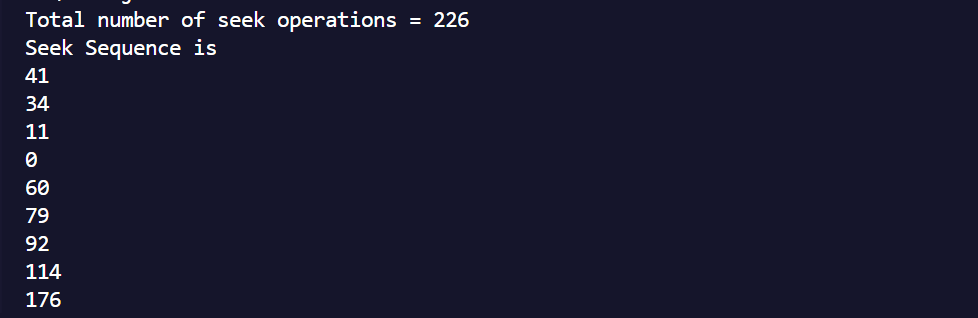
    String direction = "left";

    SCAN(arr, head, direction);

}

}

**Output :**



**Q4)C-Scan**

**Code :**

import *java.util.\**;

*class* GFG {

*static* int size = 8;

*static* int disk\_size = 200;

*public* *static* void CSCAN(int arr[], int head)

    {

        int seek\_count = 0;

        int distance, cur\_track;

        Vector<Integer> left = new Vector<Integer>();

        Vector<Integer> right = new Vector<Integer>();

        Vector<Integer> seek\_sequence

            = new Vector<Integer>();

*// Appending end values which has*

*// to be visited before reversing*

*// the direction*

        left.add(0);

        right.add(disk\_size - 1);

*// Tracks on the left of the*

*// head will be serviced when*

*// once the head comes back*

*// to the beginning (left end).*

        for (int i = 0; i < size; i++) {

            if (arr[i] < head)

                left.add(arr[i]);

            if (arr[i] > head)

                right.add(arr[i]);

        }

*// Sorting left and right vectors*

        Collections.sort(left);

        Collections.sort(right);

*// First service the requests*

*// on the right side of the*

*// head.*

        for (int i = 0; i < right.size(); i++) {

            cur\_track = right.get(i);

*// Appending current track to seek sequence*

            seek\_sequence.add(cur\_track);

*// Calculate absolute distance*

            distance = Math.abs(cur\_track - head);

*// Increase the total count*

            seek\_count += distance;

*// Accessed track is now new head*

            head = cur\_track;

        }

*// Once reached the right end*

*// jump to the beginning.*

        head = 0;

*// adding seek count for head returning from 199 to*

*// 0*

        seek\_count += (disk\_size - 1);

*// Now service the requests again*

*// which are left.*

        for (int i = 0; i < left.size(); i++) {

            cur\_track = left.get(i);

*// Appending current track to*

*// seek sequence*

            seek\_sequence.add(cur\_track);

*// Calculate absolute distance*

            distance = Math.abs(cur\_track - head);

*// Increase the total count*

            seek\_count += distance;

*// Accessed track is now the new head*

            head = cur\_track;

        }

        System.out.println("Total number of seek "

                           + "operations = " + seek\_count);

        System.out.println("Seek Sequence is");

        for (int i = 0; i < seek\_sequence.size(); i++) {

            System.out.println(seek\_sequence.get(i));

        }

    }

*// Driver code*

*public* *static* void main(String[] args) *throws* Exception

    {

*// Request array*

        int arr[] = { 176, 79, 34, 60, 92, 11, 41, 114 };

        int head = 50;

        System.out.println("Initial position of head: "

                           + head);

        CSCAN(arr, head);

    }

}

**Output :**

