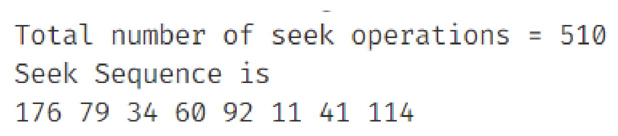
FCFS:

Code:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

void FCFS(int arr[], int head)
{
    int seek_count = 0;
    int size = 8;
    int distance, cur_track;

    for (int i = 0; i < size; i++)
    {
        cur_track = arr[i];

        // calculate absolute distance
        distance = abs(cur_track - head);

        // increase the total count
        seek_count += distance;

        // accessed track is now new head
        head = cur_track;
    }

    cout << "Total number of seek operations = "
         << seek_count << endl;

    // Seek sequence would be the same
    // as request array sequence
    cout << "Seek Sequence is" << endl;

    for (int i = 0; i < size; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main()
{
    int arr[8] = {176, 79, 34, 60, 92, 11, 41, 114};
    int head = 50;

    FCFS(arr, head);

    return 0;
}
```

Output:



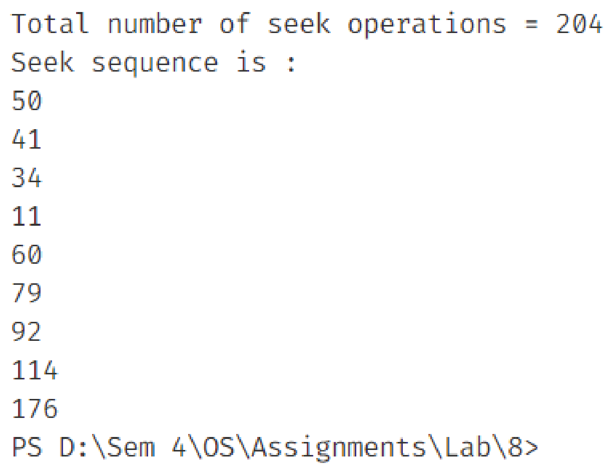
```
Total number of seek operations = 510
Seek Sequence is
176 79 34 60 92 11 41 114
```

SSTF:

Code:

```
// SSTF disk scheduling
#include <bits/stdc++.h>
using namespace std;

// Calculates difference of each
// track number with the head position
void calculatedifference(int request[], int head,
                         int diff[][2], int n)
{
    for (int i = 0; i < n; i++)
    {
        diff[i][0] = abs(head - request[i]);
    }
}

// Find unaccessed track which is
// at minimum distance from head
int findMIN(int diff[][2], int n)
{
    int index = -1;
    int minimum = 1e9;

    for (int i = 0; i < n; i++)
    {
        if (!diff[i][1] && minimum > diff[i][0])
        {
            minimum = diff[i][0];
            index = i;
        }
    }
    return index;
}

void shortestSeekTimeFirst(int request[],
                           int head, int n)
{
    if (n == 0)
    {
        return;
    }

    // Create array of objects of class node
    int diff[n][2] = {{0, 0}};

    // Count total number of seek operation
    int seekcount = 0;

    // Stores sequence in which disk access is done
    int seeksequence[n + 1] = {0};

    for (int i = 0; i < n; i++)
    {
        seeksequence[i] = head;
        calculatedifference(request, head, diff, n);
        int index = findMIN(diff, n);
        diff[index][1] = 1;

        // Increase the total count
        seekcount += diff[index][0];

        // Accessed track is now new head
        head = request[index];
    }
    seeksequence[n] = head;

    cout << "Total number of seek operations = "
         << seekcount << endl;
    cout << "Seek sequence is : "
         << "\n";

    // Print the sequence
    for (int i = 0; i <= n; i++)
    {
        cout << seeksequence[i] << "\n";
    }
}


int main()
{
    int n = 8;
    int proc[n] = {176, 79, 34, 60, 92, 11, 41, 114};

    shortestSeekTimeFirst(proc, 50, n);

    return 0;
}
```

Output:



```
Total number of seek operations = 204
Seek sequence is :
50
41
34
11
60
79
92
114
176
PS D:\Sem 4\OS\Assignments\Lab\8>
```

SCAN

Code:

```
#include <iostream>
#include <conio.h>
#include <math.h>
using namespace std;

void SCAN(int arr[], int head, int size, int disk_size, string
direction)
{
    int seek_count = 0;
    int distance, cur_track;
```

```
vector<int> left, right;
vector<int> seek_sequence;

if (direction == "left")
    left.push_back(0);
else if (direction == "right")
    right.push_back(disk_size - 1);

for (int i = 0; i < size; i++)
{
    if (arr[i] < head)
        left.push_back(arr[i]);
    if (arr[i] > head)
        right.push_back(arr[i]);
}

std::sort(left.begin(), left.end());
std::sort(right.begin(), right.end());

int run = 2;
while (run--)
{
    if (direction == "left")
    {
        for (int i = left.size() - 1; i >= 0; i--)
        {
            cur_track = left[i];

            seek_sequence.push_back(cur_track);

            distance = abs(cur_track - head);

            seek_count += distance;

            head = cur_track;
        }
        direction = "right";
    }
    else if (direction == "right")
    {
        for (int i = 0; i < right.size(); i++)
        {
            cur_track = right[i];

            seek_sequence.push_back(cur_track);

            distance = abs(cur_track - head);
```

```cpp
                seek_count += distance;

                head = cur_track;
            }
            direction = "left";
        }
    }

    cout << "Total number of seek operations = "
         << seek_count << endl;

    cout << "Seek Sequence is" << endl;

    for (int i = 0; i < seek_sequence.size(); i++)
    {
        cout << seek_sequence[i] << endl;
    }
}

int main()
{
    int size, disk_size, head;
    string direction;
    cout << "Enter the size of the request array: ";
    cin >> size;
    int arr[size];
    cout << "Enter the request array elements: ";
    for (int i = 0; i < size; i++)
    {
        cin >> arr[i];
    }
    cout << "Enter the head position: ";
    cin >> head;
    cout << "Enter the disk size: ";
    cin >> disk_size;
    cout << "Enter the direction (left or right): ";
    cin >> direction;

    SCAN(arr, head, size, disk_size, direction);

    return 0;
}
```

C-SCAN

Code:

```cpp
// C++ program to demonstrate
// C-SCAN Disk Scheduling algorithm
#include <bits/stdc++.h>
using namespace std;


int disk_size = 200;

void CSCAN(int arr[], int head)
{
    int size = 8;
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;

    // appending end values
    // which has to be visited
    // before reversing the direction
    left.push_back(0);
    right.push_back(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.push_back(arr[i]);
        if (arr[i] > head)
            right.push_back(arr[i]);
    }

    // sorting left and right vectors
    std::sort(left.begin(), left.end());
    std::sort(right.begin(), right.end());

    // first service the requests
    // on the right side of the
    // head.
    for (int i = 0; i < right.size(); i++)
    {
        cur_track = right[i];
        // appending current track to seek sequence
        seek_sequence.push_back(cur_track);

        // calculate absolute distance
        distance = 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[i];

        // appending current track to seek sequence
        seek_sequence.push_back(cur_track);

        // calculate absolute distance
        distance = abs(cur_track - head);

        // increase the total count
        seek_count += distance;

        // accessed track is now the new head
        head = cur_track;
    }

    cout << "Total number of seek operations = "
         << seek_count << endl;

    cout << "Seek Sequence is" << endl;

    for (int i = 0; i < seek_sequence.size(); i++)
    {
        cout << seek_sequence[i] << endl;
    }
}
```

```
int main()
{

    // request array
    int arr[8] = {176, 79, 34, 60, 92, 11, 41, 114};
    int head = 50;

    cout << "Initial position of head: " << head << endl;
    CSCAN(arr, head);

    return 0;
}
```

Output:

```
Initial position of head: 50
Total number of seek operations = 389
Seek Sequence is
60
79
92
114
176
199
0
11
34
41
```