**Disk Scheduling Algorithms**

**Experiment No: 7 Date: 01/11/23**

**Aim:** **a)** To implement First cum First Serve scheduling algorithm.

**Theory:**

The First-Come-First-Serve (FCFS) disk scheduling algorithm is one of the simplest disk scheduling algorithms. It services requests in the order they arrive. The head moves to each request position in the order they are requested, and the total head movement is calculated as the sum of the absolute differences between consecutive requests.

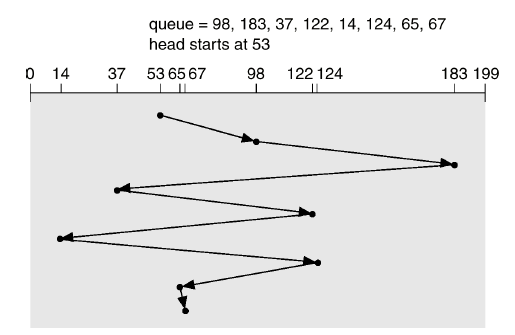
Algorithm:

1. Request queue: A sequence of disk track numbers representing the order in which the requests are received.
2. Initial head position: The current position of the disk head.
3. The disk head moves to the first request in the queue.
4. For each subsequent request in the queue, the disk head moves to that request's track.
5. The total head movement is the sum of the absolute differences between consecutive request tracks.
6. Output:

The order in which the disk head moves.

The total head movement.

Example:



Total Head Movement: 640

**CODE:**

#include<bits/stdc++.h>

using namespace std;

void FCFS(int n, int arr[], int head) {

int distance = 0;

// Traverse the array and keep adding the absolute difference to the distance to get the head movements

for (int i = 0; i < n - 1; i++) {

distance += abs(arr[i] - arr[i + 1]);

}

// Add the initial head movement to the first request

distance += abs(head - arr[0]);

cout << "The head movements are: " << distance;

}

int main() {

// Take the seek sequence

int size, head;

cout << "Enter the size of the request array: ";

cin >> size;

int seek[size];

cout << "Enter the values in the request queue: ";

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

cin >> seek[i];

}

cout << "Enter the address of the head: ";

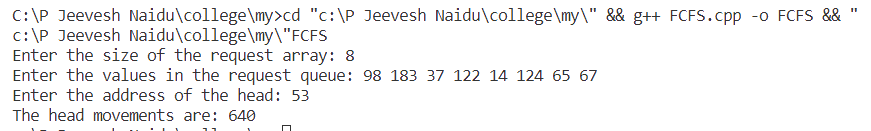
cin >> head;

FCFS(size, seek, head);

return 0;

}

**OUTPUT:**

****

**Conclusion:**

The First Cum First Serve Disk scheduling algorithm was successfully implemented in this experiment

**Disk Scheduling Algorithms**

**Experiment No: 7 Date: 01/11/23**

**Aim:** **b)** To implement SSTF Disk scheduling algorithm.

**Theory:**

The Shortest Seek Time First (SSTF) disk scheduling algorithm selects the request with the shortest seek time from the current head position. It aims to minimize the seek time by always selecting the request that is closest to the current head position.

SSTF disk scheduling algorithm:

1. Input:

Request queue: A sequence of disk track numbers representing the order in which the requests are received.

Initial head position: The current position of the disk head.

1. Procedure:

Find the request in the queue that is closest to the current head position.

Move the head to that request's track.

Remove the processed request from the queue.

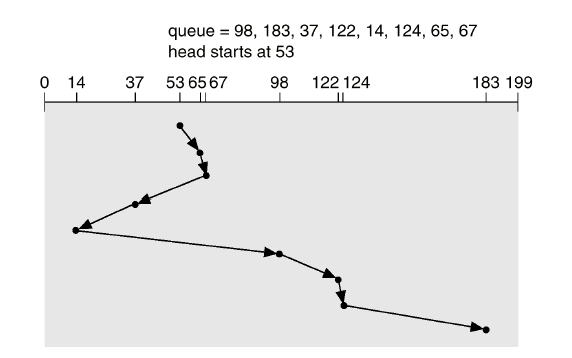
Repeat the process until all requests are processed.

1. Output:

The order in which the disk head moves.

The total head movement.

Example:



Total Head movement = 236

**CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

#include <climits>

using namespace std;

int sstf(vector<int> &requests, int current\_head) {

int total\_seek\_time = 0;

int head = current\_head;

vector<int> seekSequence;

while (!requests.empty()) {

int min\_seek = INT\_MAX;

int min\_index = -1;

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

int seek = abs(current\_head - requests[i]);

if (seek < min\_seek) {

min\_seek = seek;

min\_index = i;

}

}

seekSequence.push\_back(requests[min\_index]);

total\_seek\_time += min\_seek;

current\_head = requests[min\_index];

requests.erase(requests.begin() + min\_index);

}

// Prints the seek sequence

cout << "Seek sequence is: " << head << "-->";

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

cout << seekSequence[i] << "-->";

if (i == seekSequence.size() - 1) {

cout << seekSequence[i];

}

}

cout << endl;

return total\_seek\_time;

}

int main() {

vector<int> requests;

int request\_count;

int current\_head;

cout << "Enter the number of requests: ";

cin >> request\_count;

cout << "Enter the requests: ";

for (int i = 0; i < request\_count; i++) {

int request;

cin >> request;

requests.push\_back(request);

}

cout << "Enter the current position of the disk head: ";

cin >> current\_head;

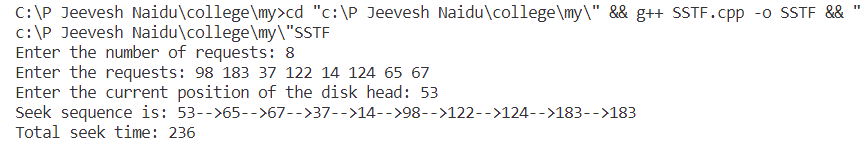
int seek\_time = sstf(requests, current\_head);

cout << "Total seek time: " << seek\_time << endl;

return 0;

}

**OUTPUT:**

****

**Conclusion:**

The SSTF Disk Scheduling Algorithm was successfully implemented in this experiment

**Disk Scheduling Algorithms**

**Experiment No: 7 Date: 08/11/23**

**Aim:** **c)** To implement SCAN Disk scheduling algorithm.

**Theory:**

The SCAN disk scheduling algorithm, also known as the Elevator algorithm, works by moving the disk arm towards one end of the disk, serving all the requests in that direction, and then moving towards the other end, serving requests in that direction.

The SCAN disk scheduling algorithm:

1. Input:

Request queue: A sequence of disk track numbers representing the order in which the requests are received.

Initial head position: The current position of the disk head.

1. Procedure:

Move the disk arm towards one end of the disk (either towards the higher-numbered tracks or the lower-numbered tracks).

While moving in one direction, serve all the requests in that direction.

When reaching the end, change the direction and serve requests in the opposite direction.

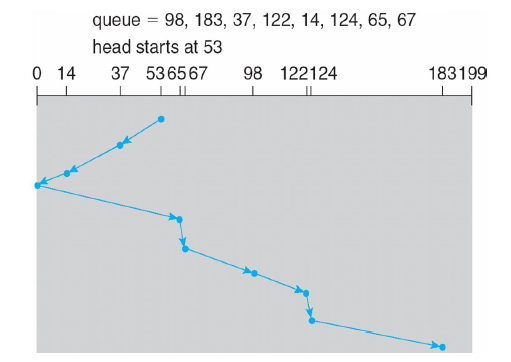
Repeat this process until all requests are processed.

1. Output:

The order in which the disk head moves.

The total head movement.

Example:



Total head movement = 236

**CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int scan(vector<int> &rq, int current\_head) {

int head = current\_head, distance = 0;

vector<int> seq;

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

auto it = lower\_bound(rq.begin(), rq.end(), head);

int midPos;

int var = 1;

while (var == 1) {

if (head > \*it) {

midPos = it - rq.begin();

var = 0;

} else {

--it;

midPos = it - rq.begin();

}

}

for (int i = midPos; i >= 0; i--) {

distance += abs(rq[i] - current\_head);

if (i == 0) {

distance += rq[0];

current\_head = 0;

seq.push\_back(rq[i]);

seq.push\_back(0);

} else {

current\_head = rq[i];

seq.push\_back(rq[i]);

}

}

for (int i = midPos + 1; i < rq.size(); i++) {

distance += abs(rq[i] - current\_head);

current\_head = rq[i];

seq.push\_back(rq[i]);

}

cout << "Seek sequence is: " << head << "-->";

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

if (i == seq.size() - 1) {

cout << seq[i];

} else {

cout << seq[i] << "-->";

}

}

cout << endl; // Add this line to print a newline

cout << "Total seek distance: " << distance;

return distance; // Return the total seek distance

}

int main() {

vector<int> requests;

int request\_count;

int current\_head;

cout << "Enter the number of requests: ";

cin >> request\_count;

cout << "Enter the requests: ";

for (int i = 0; i < request\_count; i++) {

int request;

cin >> request;

requests.push\_back(request);

}

cout << "Enter the current position of the disk head: ";

cin >> current\_head;

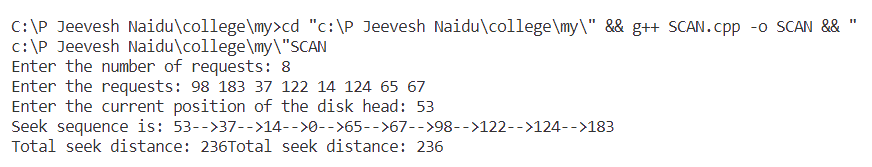
int seek\_distance = scan(requests, current\_head);

cout << "Total seek distance: " << seek\_distance << endl;

return 0;

}

**OUTPUT:**

****

**Conclusion:**

The SCAN Disk Scheduling Algorithm was successfully implemented in this experiment.

**Disk Scheduling Algorithms**

**Experiment No: 7 Date: 08/11/23**

**Aim:** **d)** To implement C-SCAN Disk scheduling algorithm.

**Theory:**

The CSCAN (Circular SCAN) disk scheduling algorithm is an improvement over the SCAN algorithm. CSCAN works by moving the disk arm towards one end of the disk, serving all the requests in that direction. When reaching the end, it immediately jumps to the other end without servicing any requests in between, and continues serving requests in the same direction. This process repeats until all requests are processed.

CSCAN disk scheduling algorithm:

1. Input:

Request queue: A sequence of disk track numbers representing the order in which the requests are received.

Initial head position: The current position of the disk head.

1. Procedure:

Move the disk arm towards one end of the disk (either towards the higher-numbered tracks or the lower-numbered tracks).

While moving in one direction, serve all the requests in that direction.

When reaching the end, jump to the other end without serving any requests in between.

Continue serving requests in the same direction.

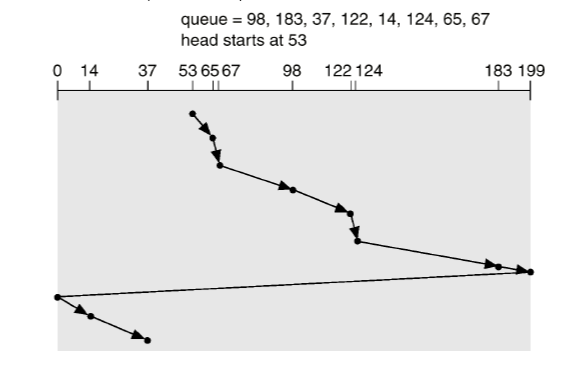
Repeat this process until all requests are processed.

1. Output:

The order in which the disk head moves.

The total head movement.

Example:



Total head movement = 382

**CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int scan(vector<int> &rq, int current\_head, int limit) {

int head = current\_head, distance = 0;

vector<int> seq;

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

auto it = lower\_bound(rq.begin(), rq.end(), head);

int midPos;

int var = 1;

while (var == 1) {

if (head < \*it) {

midPos = it - rq.begin();

var = 0;

} else {

++it;

midPos = it - rq.begin();

}

}

for (int i = midPos; i < rq.size(); i++) {

distance += abs(rq[i] - current\_head);

if (i == rq.size() - 1) {

distance += abs(rq[rq.size() - 1] - limit) + limit;

current\_head = 0;

seq.push\_back(rq[i]);

seq.push\_back(limit);

seq.push\_back(0);

} else {

current\_head = rq[i];

seq.push\_back(rq[i]);

}

}

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

distance += abs(rq[i] - current\_head);

current\_head = rq[i];

seq.push\_back(rq[i]);

}

cout << "Seek sequence is: " << head << "-->";

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

if (i == seq.size() - 1) {

cout << seq[i];

} else {

cout << seq[i] << "-->";

}

}

cout << endl;

return distance; // Return the total seek distance

}

int main() {

vector<int> requests;

int request\_count;

int current\_head;

int limit;

cout << "Enter the number of requests: ";

cin >> request\_count;

cout << "Enter the requests: ";

for (int i = 0; i < request\_count; i++) {

int request;

cin >> request;

requests.push\_back(request);

}

cout << "Enter the current position of the disk head: ";

cin >> current\_head;

cout << "Enter the limit of the memory: ";

cin >> limit;

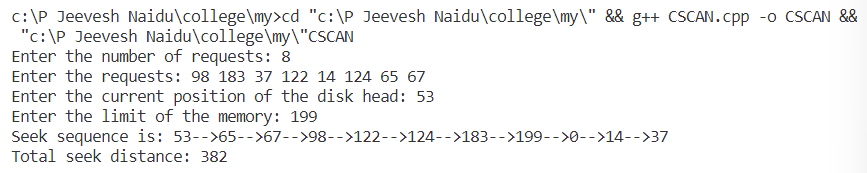
int seek\_distance = scan(requests, current\_head, limit);

cout << "Total seek distance: " << seek\_distance << endl;

return 0;

}

**OUTPUT:**

****

**Conclusion:**

The C-Scan Disk Scheduling Algorithm was successfully implemented in this experiment.

**Disk Scheduling Algorithms**

**Experiment No: 7 Date: 15/11/23**

**Aim:** **e)** To implement LOOK Disk scheduling algorithm.

**Theory:**

The LOOK algorithm is similar to the SCAN algorithm, but it does not go all the way to the end of the disk in both directions. Instead, it reverses direction when there are no more requests in the current direction.

LOOK disk scheduling algorithm:

1. Input:

Request queue: A sequence of disk track numbers representing the order in which the requests are received.

Initial head position: The current position of the disk head.

1. Procedure:

Move the disk arm towards one end of the disk (either towards the higher-numbered tracks or the lower-numbered tracks).

While moving in one direction, serve all the requests in that direction.

When there are no more requests in the current direction, reverse the direction.

Continue serving requests in the opposite direction.

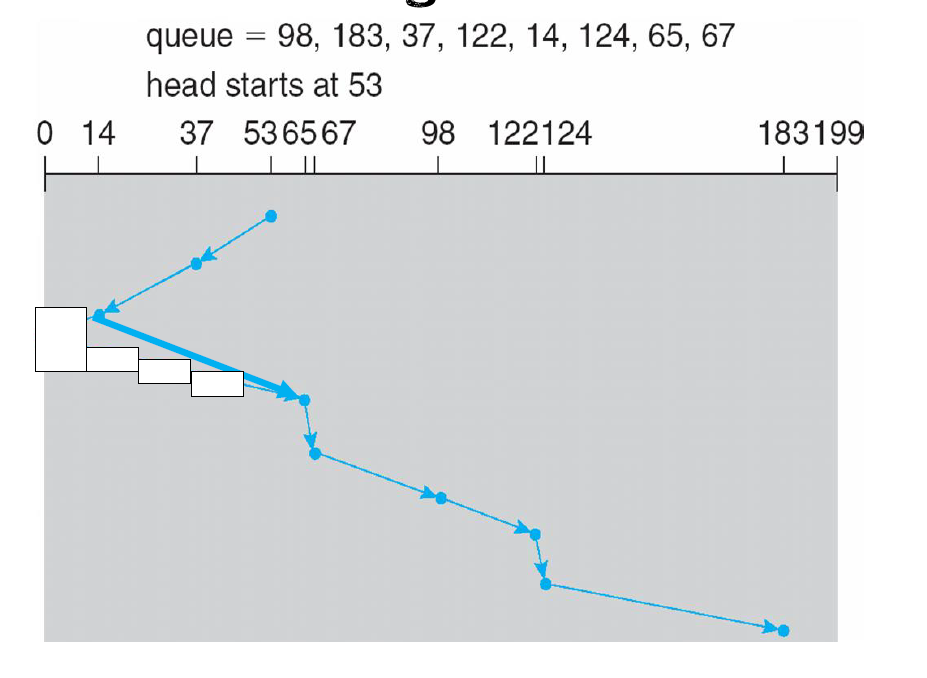
Repeat this process until all requests are processed.

1. Output:

The order in which the disk head moves.

The total head movement.

Example:



Total Head movement = 208

**CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int look(vector<int> &rq, int current\_head) {

int head = current\_head, distance = 0;

vector<int> seq;

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

auto it = lower\_bound(rq.begin(), rq.end(), head);

int midPos;

int var = 1;

while (var == 1) {

if (head > \*it) {

midPos = it - rq.begin();

var = 0;

} else {

--it;

midPos = it - rq.begin();

}

}

for (int i = midPos; i >= 0; i--) {

distance += abs(rq[i] - current\_head);

current\_head = rq[i];

seq.push\_back(rq[i]);

}

for (int i = midPos + 1; i < rq.size(); i++) {

distance += abs(rq[i] - current\_head);

current\_head = rq[i];

seq.push\_back(rq[i]);

}

cout << "Seek sequence is: " << head << "-->";

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

if (i == seq.size() - 1) {

cout << seq[i];

} else {

cout << seq[i] << "-->";

}

}

cout << endl;

return distance; // Return the total seek distance

}

int main() {

vector<int> requests;

int request\_count;

int current\_head;

cout << "Enter the number of requests: ";

cin >> request\_count;

cout << "Enter the requests: ";

for (int i = 0; i < request\_count; i++) {

int request;

cin >> request;

requests.push\_back(request);

}

cout << "Enter the current position of the disk head: ";

cin >> current\_head;

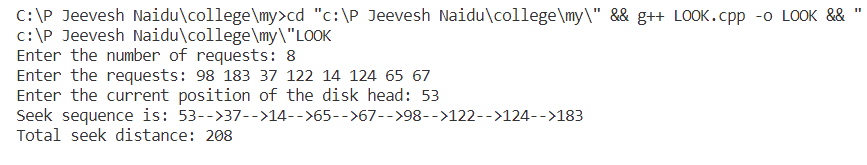
int seek\_distance = look(requests, current\_head);

cout << "Total seek distance: " << seek\_distance << endl;

return 0;

}

**OUTPUT:**

****

**Conclusion:**

The LOOK Disk Scheduling Algorithm was successfully implemented in this experiment.

**Disk Scheduling Algorithms**

**Experiment No: 7 Date: 15/11/23**

**Aim:** **f)** To implement C-LOOK Disk scheduling algorithm.

**Theory:**

The C-LOOK (Circular LOOK) disk scheduling algorithm is a variation of the LOOK algorithm. It services requests in one direction until there are no more requests in that direction, and then it jumps to the other end without servicing any requests in between. This process repeats until all requests are processed.

C-LOOK disk scheduling algorithm:

1. Input:

Request queue: A sequence of disk track numbers representing the order in which the requests are received.

Initial head position: The current position of the disk head.

1. Procedure:

Move the disk arm towards one end of the disk (either towards the higher-numbered tracks or the lower-numbered tracks).

While moving in one direction, serve all the requests in that direction.

When there are no more requests in the current direction, jump to the other end without serving any requests in between.

Continue serving requests in the same direction.

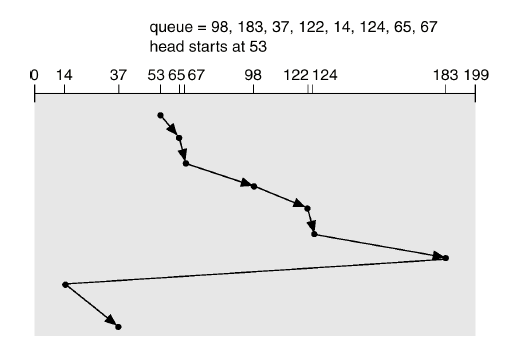
Repeat this process until all requests are processed.

1. Output:

The order in which the disk head moves.

The total head movement.

Example:



Total head movement = 322

**CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int scan(vector<int> &rq, int current\_head, int limit) {

int head = current\_head, distance = 0;

vector<int> seq;

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

auto it = lower\_bound(rq.begin(), rq.end(), head);

int mid = \*it;

int midPos;

int var = 1;

while (var == 1) {

if (head < \*it) {

midPos = it - rq.begin();

var = 0;

} else {

++it;

midPos = it - rq.begin();

}

}

for (int i = midPos; i < rq.size(); i++) {

distance += abs(rq[i] - current\_head);

current\_head = rq[i];

seq.push\_back(rq[i]);

}

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

distance += abs(rq[i] - current\_head);

current\_head = rq[i];

seq.push\_back(rq[i]);

}

cout << "Seek sequence is: " << head << "-->";

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

if (i == seq.size() - 1) {

cout << seq[i];

} else {

cout << seq[i] << "-->";

}

}

cout << endl;

return distance; // Return the total seek distance

}

int main() {

vector<int> requests;

int request\_count;

int current\_head;

int limit;

cout << "Enter the number of requests: ";

cin >> request\_count;

cout << "Enter the requests: ";

for (int i = 0; i < request\_count; i++) {

int request;

cin >> request;

requests.push\_back(request);

}

cout << "Enter the current position of the disk head: ";

cin >> current\_head;

cout << "Enter the limit of the memory: ";

cin >> limit;

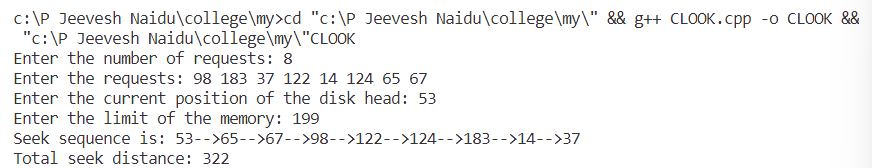
int seek\_distance = scan(requests, current\_head, limit);

cout << "Total seek distance: " << seek\_distance << endl;

return 0;

}

**OUTPUT:**

****

**Conclusion:**

The C-LOOK Disk Scheduling Algorithm was successfully implemented in this experiment.