Q9. C program to implement Banker’s Algorithm

Source Code

#include <stdio.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int available[MAX\_RESOURCES];

int max[MAX\_PROCESSES][MAX\_RESOURCES];

int allocated[MAX\_PROCESSES][MAX\_RESOURCES];

int need[MAX\_PROCESSES][MAX\_RESOURCES];

int isSafe(int processes, int resources) {

int work[MAX\_RESOURCES];

int finish[MAX\_PROCESSES] = {0};

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

work[i] = available[i];

}

int count = 0;

while (count < processes) {

int found = 0;

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

if (!finish[i]) {

int j;

for (j = 0; j < resources; j++) {

if (need[i][j] > work[j]) {

break;

}

}

if (j == resources) {

for (int k = 0; k < resources; k++) {

work[k] += allocated[i][k];

}

finish[i] = 1;

found = 1;

count++;

}

}

}

if (!found) {

return 0;

}

}

return 1;

}

void requestResources(int processes, int resources, int process, int request[]) {

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

if (request[i] > need[process][i] || request[i] > available[i]) {

printf("Invalid request. The request exceeds the maximum need or available resources.\n");

return;

}

}

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

allocated[process][i] += request[i];

available[i] -= request[i];

need[process][i] -= request[i];

}

if (isSafe(processes, resources)) {

printf("Request granted. The system is in a safe state.\n");

} else {

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

allocated[process][i] -= request[i];

available[i] += request[i];

need[process][i] += request[i];

}

printf("Request denied. The system would be in an unsafe state.\n");

}

}

void releaseResources(int resources, int process, int release[]) {

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

if (release[i] > allocated[process][i]) {

printf("Invalid release. The release exceeds the allocated resources.\n");

return;

}

}

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

allocated[process][i] -= release[i];

available[i] += release[i];

}

printf("Resources released. The system is in a safe state.\n");

}

int main() {

int processes, resources;

printf("Enter the number of processes: ");

scanf("%d", &processes);

printf("Enter the number of resources: ");

scanf("%d", &resources);

printf("Enter the maximum resources matrix:\n");

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

printf("Process %d: ", i);

for (int j = 0; j < resources; j++) {

scanf("%d", &max[i][j]);

}

}

printf("Enter the allocated resources matrix:\n");

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

printf("Process %d: ", i);

for (int j = 0; j < resources; j++) {

scanf("%d", &allocated[i][j]);

need[i][j] = max[i][j] - allocated[i][j];

}

}

printf("Enter the available resources vector:\n");

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

scanf("%d", &available[i]);

}

if (isSafe(processes, resources)) {

printf("The initial state is safe.\n");

} else {

printf("The initial state is unsafe.\n");

return 1;

}

// Demonstrate resource request and release

int process, request[MAX\_RESOURCES], release[MAX\_RESOURCES];

printf("Enter the process number requesting resources: ");

scanf("%d", &process);

printf("Enter the resource request (e.g., R1 R2 ...): ");

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

scanf("%d", &request[i]);

}

requestResources(processes, resources, process, request);

printf("Enter the process number releasing resources: ");

scanf("%d", &process);

printf("Enter the resource release (e.g., R1 R2 ...): ");

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

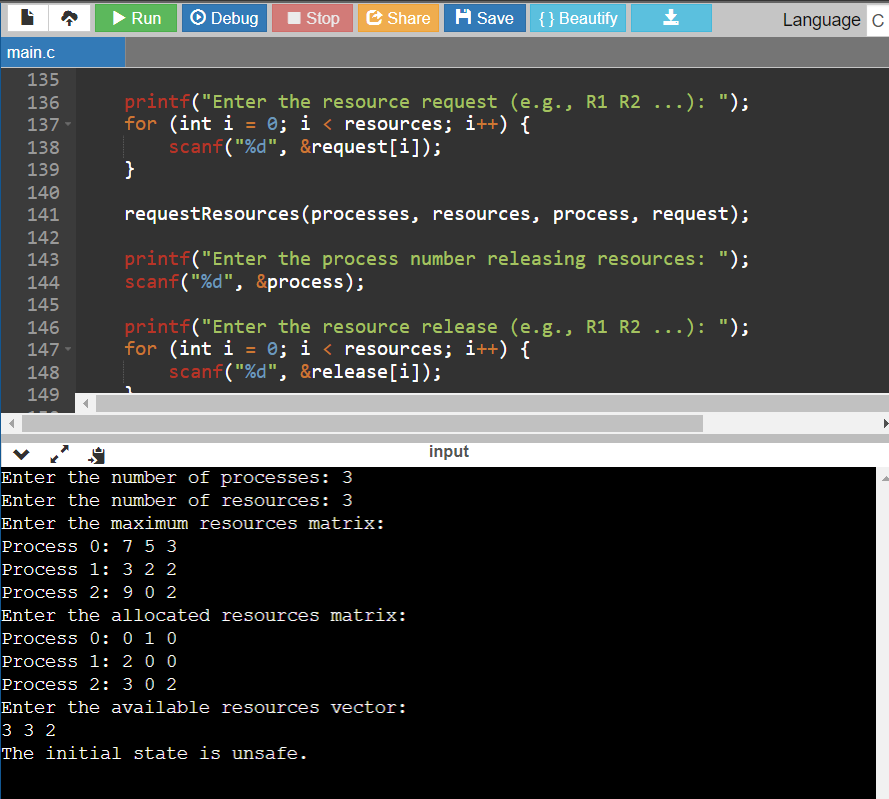
scanf("%d", &release[i]);

}

releaseResources(resources, process, release);

return 0;}

Output



Q10. C program to implement first in first out page replacement policy

Source Code

#include <stdio.h>

#define MAX\_FRAMES 3

void initializeFrames(int frames[MAX\_FRAMES]) {

for (int i = 0; i < MAX\_FRAMES; i++) {

frames[i] = -1; // -1 indicates an empty frame

}

}

void printFrames(int frames[MAX\_FRAMES]) {

printf("Frames: ");

for (int i = 0; i < MAX\_FRAMES; i++) {

if (frames[i] == -1) {

printf("[ ] ");

} else {

printf("[%d] ", frames[i]);

}

}

printf("\n");

}

int isPageInFrames(int frames[MAX\_FRAMES], int page) {

for (int i = 0; i < MAX\_FRAMES; i++) {

if (frames[i] == page) {

return 1; // Page is already in frames

}

}

return 0; // Page is not in frames

}

void fifoPageReplacement(int frames[MAX\_FRAMES], int page, int \*nextFrameIndex) {

frames[\*nextFrameIndex] = page;

\*nextFrameIndex = (\*nextFrameIndex + 1) % MAX\_FRAMES;}

int main() {

int frames[MAX\_FRAMES];

initializeFrames(frames);

int pageSequence[] = {0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4};

int pageSequenceSize = sizeof(pageSequence) / sizeof(pageSequence[0]);

int pageFaults = 0;

int nextFrameIndex = 0;

printf("Page Replacement using FIFO:\n");

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

int currentPage = pageSequence[i];

if (!isPageInFrames(frames, currentPage)) {

printf("Page %d caused a page fault. ", currentPage);

fifoPageReplacement(frames, currentPage, &nextFrameIndex);

pageFaults++;

} else {

printf("Page %d is already in memory. ", currentPage);}

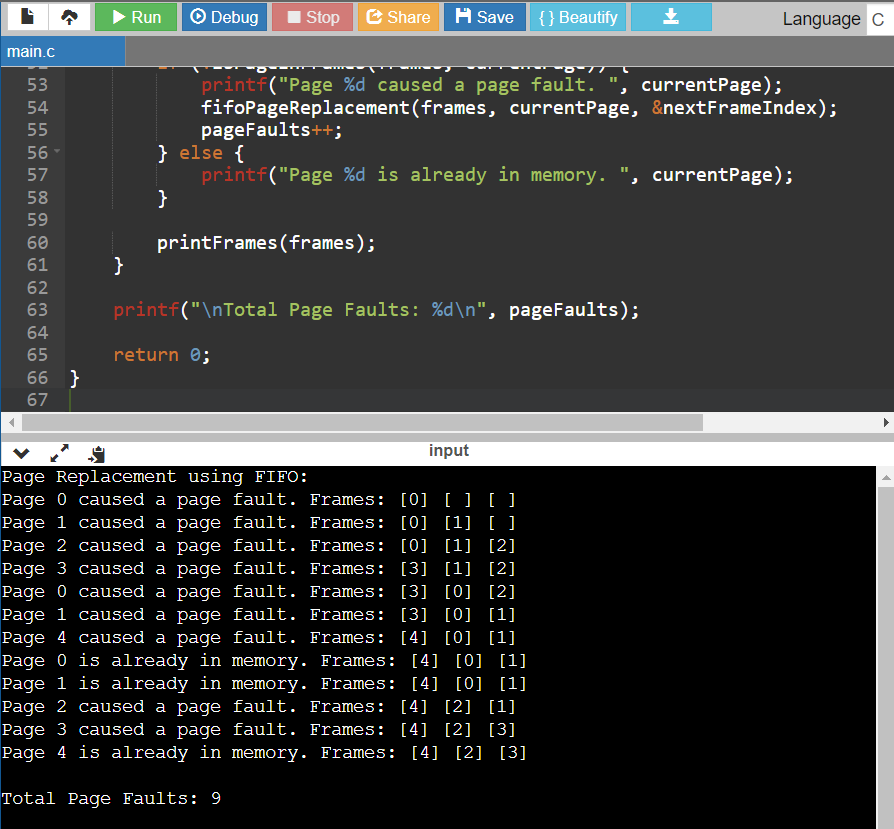
printFrames(frames);}

printf("\nTotal Page Faults: %d\n", pageFaults);

return 0;

}

Output



Q11. C program to implement least recently used page replacement policy

Source Code

#include <stdio.h>

#define MAX\_FRAMES 3

void initializeFrames(int frames[MAX\_FRAMES]) {

for (int i = 0; i < MAX\_FRAMES; i++) {

frames[i] = -1; // -1 indicates an empty frame

}

}

void printFrames(int frames[MAX\_FRAMES]) {

printf("Frames: ");

for (int i = 0; i < MAX\_FRAMES; i++) {

if (frames[i] == -1) {

printf("[ ] ");

} else {

printf("[%d] ", frames[i]);

}

}

printf("\n");

}

int isPageInFrames(int frames[MAX\_FRAMES], int page) {

for (int i = 0; i < MAX\_FRAMES; i++) {

if (frames[i] == page) {

return 1; // Page is already in frames

}

}

return 0; // Page is not in frames

}

int getLRUPage(int pageOrder[MAX\_FRAMES]) {

return pageOrder[MAX\_FRAMES - 1];

}

void updatePageOrder(int pageOrder[MAX\_FRAMES], int currentPage) {

// Move the current page to the front of the page order

for (int i = 0; i < MAX\_FRAMES; i++) {

if (pageOrder[i] == currentPage) {

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

pageOrder[j] = pageOrder[j - 1];

}

pageOrder[0] = currentPage;

break;

}

}

}

void lruPageReplacement(int frames[MAX\_FRAMES], int pageOrder[MAX\_FRAMES], int page) {

int leastRecentlyUsedPage = getLRUPage(pageOrder);

for (int i = 0; i < MAX\_FRAMES; i++) {

if (frames[i] == leastRecentlyUsedPage) {

frames[i] = page;

break;

}

}

updatePageOrder(pageOrder, page);

}

int main() {

int frames[MAX\_FRAMES];

initializeFrames(frames);

int pageSequence[] = {0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4};

int pageSequenceSize = sizeof(pageSequence) / sizeof(pageSequence[0]);

int pageOrder[MAX\_FRAMES];

for (int i = 0; i < MAX\_FRAMES; i++) {

pageOrder[i] = -1; // Initialize page order

}

int pageFaults = 0;

printf("Page Replacement using LRU:\n");

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

int currentPage = pageSequence[i];

if (!isPageInFrames(frames, currentPage)) {

printf("Page %d caused a page fault. ", currentPage);

if (pageFaults < MAX\_FRAMES) {

frames[pageFaults] = currentPage;

pageOrder[pageFaults] = currentPage;

} else {

lruPageReplacement(frames, pageOrder, currentPage);

}

pageFaults++;

} else {

printf("Page %d is already in memory. ", currentPage);

updatePageOrder(pageOrder, currentPage);

}

printFrames(frames);

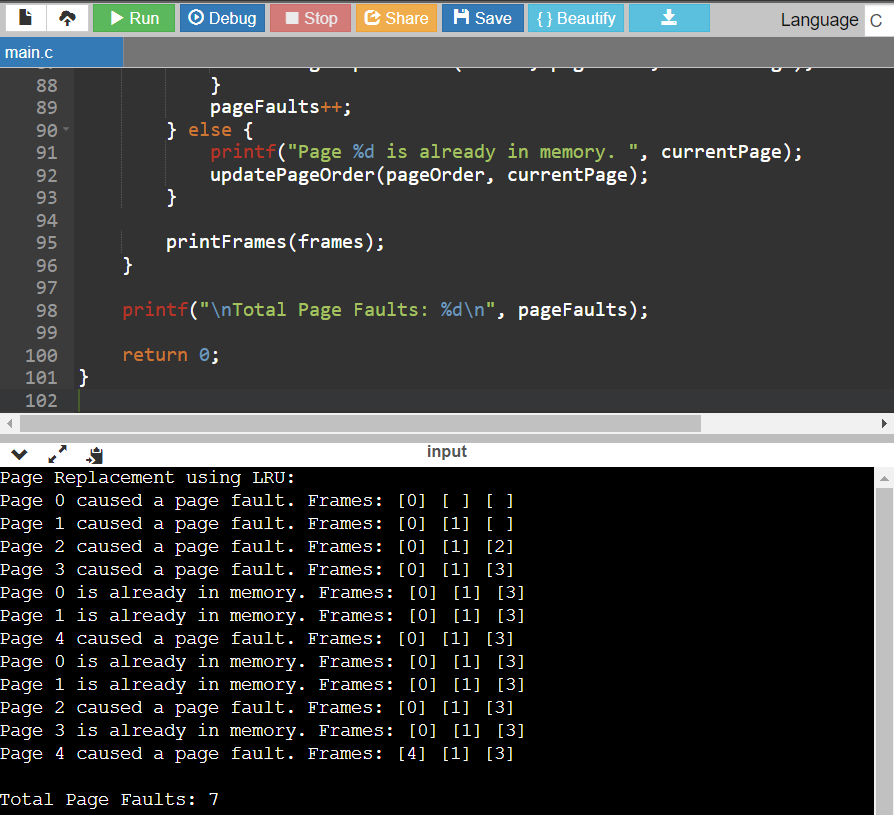
}

printf("\nTotal Page Faults: %d\n", pageFaults);

return 0;

}

Output



\

Q12. C program to implement FCFS Disk Scheduling Algorithm

Source Code

#include <stdio.h>

#include <stdlib.h>

void calculateSeekTime(int requestSequence[], int numRequests, int initialHeadPosition) {

int seekTime = 0;

int currentHeadPosition = initialHeadPosition;

printf("Seek Sequence: %d", currentHeadPosition);

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

int distance = abs(requestSequence[i] - currentHeadPosition);

seekTime += distance;

currentHeadPosition = requestSequence[i];

printf(" -> %d", currentHeadPosition);

}

printf("\nTotal Seek Time: %d\n", seekTime);

}

int main() {

int numRequests;

int initialHeadPosition;

printf("Enter the number of requests: ");

scanf("%d", &numRequests);

int \*requestSequence = (int \*)malloc(numRequests \* sizeof(int));

if (requestSequence == NULL) {

fprintf(stderr, "Memory allocation failed.\n");

return 1; // Exit with an error code

}

printf("Enter the request sequence:\n");

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

scanf("%d", &requestSequence[i]);

}

printf("Enter the initial head position: ");

scanf("%d", &initialHeadPosition);

calculateSeekTime(requestSequence, numRequests, initialHeadPosition);

free(requestSequence);

return 0;

}

Output



Q13. C program to implement the SSTF Disk Scheduling Algorithm

Source Code

#include <stdio.h>

#include <stdlib.h>

#include <limits.h> // Include the header file for INT\_MAX

// Function to calculate seek time using SSTF algorithm

void calculateSeekTime(int requestSequence[], int numRequests, int initialHeadPosition) {

int seekTime = 0;

int currentHeadPosition = initialHeadPosition;

int visited[numRequests];

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

visited[i] = 0; // Initialize all requests as not visited

}

printf("Seek Sequence: %d", currentHeadPosition);

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

int minDistance = INT\_MAX; // Use INT\_MAX from <limits.h>

int nextRequest = -1;

// Find the request with the shortest seek time

for (int j = 0; j < numRequests; j++) {

if (!visited[j]) {

int distance = abs(requestSequence[j] - currentHeadPosition);

if (distance < minDistance) {

minDistance = distance;

nextRequest = j;

}

}

}

visited[nextRequest] = 1; // Mark the request as visited

seekTime += minDistance;

currentHeadPosition = requestSequence[nextRequest];

printf(" -> %d", currentHeadPosition);

}

printf("\nTotal Seek Time: %d\n", seekTime);

}

int main() {

int numRequests;

int initialHeadPosition;

printf("Enter the number of requests: ");

scanf("%d", &numRequests);

int \*requestSequence = (int \*)malloc(numRequests \* sizeof(int));

if (requestSequence == NULL) {

fprintf(stderr, "Memory allocation failed.\n");

return 1; // Exit with an error code

}

printf("Enter the request sequence:\n");

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

scanf("%d", &requestSequence[i]);

}

printf("Enter the initial head position: ");

scanf("%d", &initialHeadPosition);

calculateSeekTime(requestSequence, numRequests, initialHeadPosition);

free(requestSequence);

return 0;

}

Output



Q14. C program to implement SCAN Disk Scheduling Algorithm

Source Code

#include <stdio.h>

#include <stdlib.h>

void calculateSeekTime(int requestSequence[], int numRequests, int initialHeadPosition, int direction) {

int seekTime = 0;

int currentHeadPosition = initialHeadPosition;

printf("Seek Sequence: %d", currentHeadPosition);

if (direction == 1) { // Move towards higher cylinder numbers

// Go to the end of the disk

for (int i = currentHeadPosition; i <= 199; i++) {

printf(" -> %d", i);

seekTime += abs(i - currentHeadPosition);

currentHeadPosition = i;

}

// Go to the beginning of the disk

for (int i = 199; i >= 0; i--) {

printf(" -> %d", i);

seekTime += abs(i - currentHeadPosition);

currentHeadPosition = i;

}

} else { // Move towards lower cylinder numbers

// Go to the beginning of the disk

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

printf(" -> %d", i);

seekTime += abs(i - currentHeadPosition);

currentHeadPosition = i;

}

// Go to the end of the disk

for (int i = 0; i <= 199; i++) {

printf(" -> %d", i);

seekTime += abs(i - currentHeadPosition);

currentHeadPosition = i;

}

}

printf("\nTotal Seek Time: %d\n", seekTime);

}

int main() {

int numRequests;

int initialHeadPosition;

int direction;

printf("Enter the number of requests: ");

scanf("%d", &numRequests);

int \*requestSequence = (int \*)malloc(numRequests \* sizeof(int));

if (requestSequence == NULL) {

fprintf(stderr, "Memory allocation failed.\n");

return 1; // Exit with an error code

}

printf("Enter the request sequence:\n");

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

scanf("%d", &requestSequence[i]);

}

printf("Enter the initial head position: ");

scanf("%d", &initialHeadPosition);

printf("Enter the direction (1 for towards higher cylinders, 0 for towards lower cylinders): ");

scanf("%d", &direction);

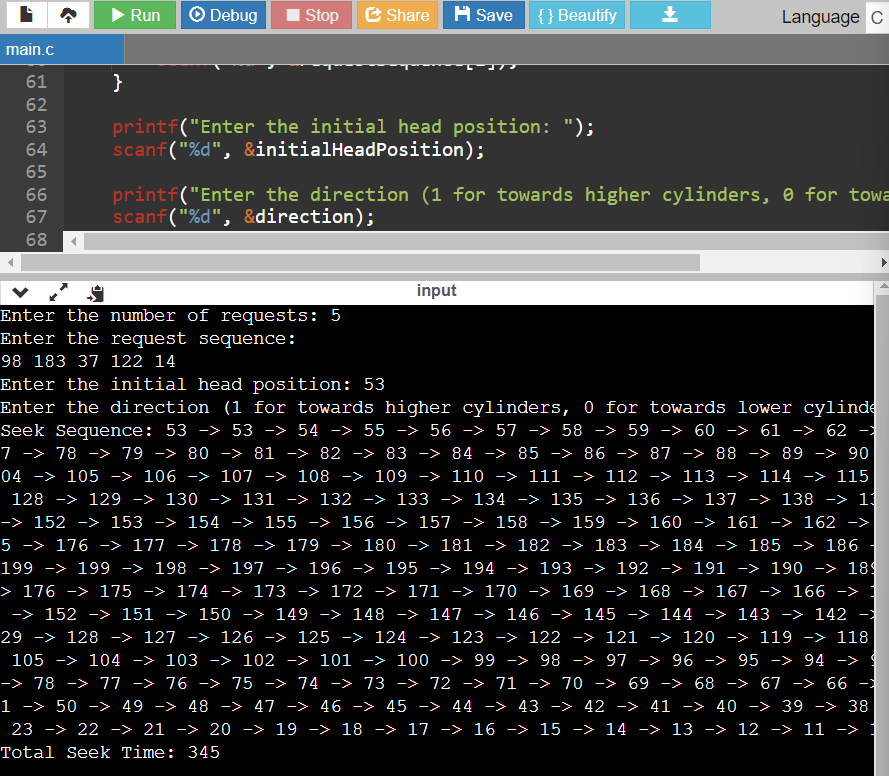
calculateSeekTime(requestSequence, numRequests, initialHeadPosition, direction);

free(requestSequence);

return 0;

}

Output



Q15. C program to implement C-SCAN Disk scheduling

Source Code

#include <stdio.h>

#include <stdlib.h>

void calculateSeekTime(int requestSequence[], int numRequests, int initialHeadPosition) {

int seekTime = 0;

int currentHeadPosition = initialHeadPosition;

printf("Seek Sequence: %d", currentHeadPosition);

// Sort the request sequence

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

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

if (requestSequence[j] > requestSequence[j + 1]) {

// Swap the requests if they are out of order

int temp = requestSequence[j];

requestSequence[j] = requestSequence[j + 1];

requestSequence[j + 1] = temp;

}

}

}

// Find the index where the current head position is located in the sorted sequence

int index = 0;

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

if (requestSequence[i] >= currentHeadPosition) {

index = i;

break;

}

}

// Go to the end of the disk

for (int i = index; i < numRequests; i++) {

printf(" -> %d", requestSequence[i]);

seekTime += abs(requestSequence[i] - currentHeadPosition);

currentHeadPosition = requestSequence[i];

}

// Go to the beginning of the disk

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

printf(" -> %d", requestSequence[i]);

seekTime += abs(requestSequence[i] - currentHeadPosition);

currentHeadPosition = requestSequence[i];

}

printf("\nTotal Seek Time: %d\n", seekTime);

}

int main() {

int numRequests;

int initialHeadPosition;

printf("Enter the number of requests: ");

scanf("%d", &numRequests);

int \*requestSequence = (int \*)malloc(numRequests \* sizeof(int));

if (requestSequence == NULL) {

fprintf(stderr, "Memory allocation failed.\n");

return 1; // Exit with an error code

}

printf("Enter the request sequence:\n");

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

scanf("%d", &requestSequence[i]);

}

printf("Enter the initial head position: ");

scanf("%d", &initialHeadPosition);

calculateSeekTime(requestSequence, numRequests, initialHeadPosition);

free(requestSequence);

return 0;

}

Output

