```c
#include <stdio.h>
#include <stdlib.h>


void sort(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] > arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}


int findClosestRequest(int requests[], int n, int head, int visited[]) {
    int minDiff = 10000;  // Large value to find the minimum difference
    int closestIdx = -1;

    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            int diff = abs(requests[i] - head);
            if (diff < minDiff) {
                minDiff = diff;
                closestIdx = i;
            }
        }
    }

    return closestIdx;
}

void SSTF(int requests[], int n, int head) {
    int visited[n];
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }

    int seekCount = 0;
    int sequence[n];

    printf("Seek sequence (SSTF): ");
    for (int i = 0; i < n; i++) {
        int closestIdx = findClosestRequest(requests, n, head, visited);
        visited[closestIdx] = 1;
        seekCount += abs(requests[closestIdx] - head);
        head = requests[closestIdx];
        sequence[i] = head;
        printf("%d ", head);
```

```c
    }

    printf("\nTotal Seek Time (SSTF): %d\n", seekCount);
}

// SCAN (Elevator Algorithm)
void SCAN(int requests[], int n, int head, int disk_size, int direction) {
    int seekCount = 0;
    int sequence[n + 2];
    int idx = 0;

    sort(requests, n);

    printf("Seek sequence (SCAN): ");

    // Service requests in the specified direction
    if (direction == 1) {  // Move towards higher
        for (int i = 0; i < n; i++) {
            if (requests[i] > head) {
                sequence[idx++] = requests[i];
                seekCount += abs(requests[i] - head);
                head = requests[i];
            }
        }
        // Go to the end of the disk
        sequence[idx++] = disk_size;
        seekCount += abs(disk_size - head);
        head = disk_size;

        // Move back in the opposite direction
        for (int i = n - 1; i >= 0; i--) {
            if (requests[i] < head) {
                sequence[idx++] = requests[i];
                seekCount += abs(requests[i] - head);
                head = requests[i];
            }
        }
    } else {  // Move towards lower
        for (int i = n - 1; i >= 0; i--) {
            if (requests[i] < head) {
                sequence[idx++] = requests[i];
                seekCount += abs(requests[i] - head);
                head = requests[i];
            }
        }
        // Go to the start of the disk (0)
        sequence[idx++] = 0;
        seekCount += head;  // Head was at the last serviced request

        // Move in the other direction
        for (int i = 0; i < n; i++) {
            if (requests[i] > head) {
```

```
                sequence[idx++] = requests[i];
                seekCount += abs(requests[i] - head);
                head = requests[i];
            }
        }
    }

    // Print the seek sequence
    for (int i = 0; i < idx; i++) {
        printf("%d ", sequence[i]);
    }

    printf("\nTotal Seek Time (SCAN): %d\n", seekCount);
}

// C-LOOK (Circular LOOK)
void C_LOOK(int requests[], int n, int head) {
    int seekCount = 0;
    int sequence[n + 1];
    int idx = 0;

    sort(requests, n);

    printf("Seek sequence (C-LOOK): ");

    // Service requests to the right of the head
    for (int i = 0; i < n; i++) {
        if (requests[i] > head) {
            sequence[idx++] = requests[i];
            seekCount += abs(requests[i] - head);
            head = requests[i];
        }
    }

    // Jump back to the beginning of the requests (smallest)
    seekCount += abs(requests[0] - head);
    head = requests[0];
    sequence[idx++] = head;

    // Service the remaining requests
    for (int i = 1; i < n; i++) {
        if (requests[i] > head) {
            sequence[idx++] = requests[i];
            seekCount += abs(requests[i] - head);
            head = requests[i];
        }
    }

    // Print the seek sequence
    for (int i = 0; i < idx; i++) {
        printf("%d ", sequence[i]);
    }
```

```c

    printf("\nTotal Seek Time (C-LOOK): %d\n", seekCount);
}

// Menu-driven program
int main() {
    int n, head, disk_size, direction, choice;

    printf("Enter the number of disk requests: ");
    scanf("%d", &n);

    int requests[n];
    printf("Enter the disk request queue: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &head);

    while (1) {
        printf("\nDisk Scheduling Algorithms:\n");
        printf("1. SSTF\n");
        printf("2. SCAN\n");
        printf("3. C-LOOK\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                SSTF(requests, n, head);
                break;
            case 2:
                printf("Enter disk size (max cylinder): ");
                scanf("%d", &disk_size);
                printf("Enter initial direction (1 for high, 0 for low): ");
                scanf("%d", &direction);
                SCAN(requests, n, head, disk_size, direction);
                break;
            case 3:
                C_LOOK(requests, n, head);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice! Please select again.\n");
        }
    }

    return 0;
}
```