Experiment No-8

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

void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

// Function to sort the array (used in SCAN and C-LOOK)
void sort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
}

// SSTF
void SSTF(int requests[], int n, int head) {
    int visited[n];
    int total_seek = 0;
    int current = head;

    for (int i = 0; i < n; i++)
        visited[i] = 0;

    printf("\nSSTF Disk Scheduling:\nOrder of Access: ");
    for (int count = 0; count < n; count++) {
        int min = 1e9, index = -1;
        for (int i = 0; i < n; i++) {
            if (!visited[i]) {
                int distance = abs(current - requests[i]);
                if (distance < min) {
                    min = distance;
                    index = i;
                }
            }
        }

        visited[index] = 1;
        total_seek += abs(current - requests[index]);
        current = requests[index];
        printf("%d ", current);
    }

    printf("\nTotal Seek Time: %d\n", total_seek);
    printf("Average Seek Time: %.2f\n", (float)total_seek / n);
```

```
}

// SCAN
void SCAN(int requests[], int n, int head, int disk_size) {
    int total_seek = 0;
    int current = head;

    // Sort the requests
    sort(requests, n);

    printf("\nSCAN Disk Scheduling:\nOrder of Access: ");

    // Find the split point
    int i;
    for (i = 0; i < n; i++)
        if (requests[i] > head)
            break;

    // Move right (away from spindle)
    for (int j = i; j < n; j++) {
        printf("%d ", requests[j]);
        total_seek += abs(current - requests[j]);
        current = requests[j];
    }

    // Hit the end of the disk
    if (current != disk_size - 1) {
        total_seek += abs(current - (disk_size - 1));
        current = disk_size - 1;
    }

    // Move back towards 0, servicing remaining requests
    for (int j = i - 1; j >= 0; j--) {
        printf("%d ", requests[j]);
        total_seek += abs(current - requests[j]);
        current = requests[j];
    }

    printf("\nTotal Seek Time: %d\n", total_seek);
    printf("Average Seek Time: %.2f\n", (float)total_seek / n);
}

// C-LOOK
void CLOOK(int requests[], int n, int head) {
    int total_seek = 0;
    int current = head;

    // Sort the requests
    sort(requests, n);

    printf("\nC-LOOK Disk Scheduling:\nOrder of Access: ");
```

```
    // Find the split point
    int i;
    for (i = 0; i < n; i++)
        if (requests[i] > head)
            break;

    // Move right (away from spindle)
    for (int j = i; j < n; j++) {
        printf("%d ", requests[j]);
        total_seek += abs(current - requests[j]);
        current = requests[j];
    }

    // Jump to the lowest request
    if (i != 0) {
        total_seek += abs(current - requests[0]);
        current = requests[0];
    }

    for (int j = 0; j < i; j++) {
        printf("%d ", requests[j]);
        total_seek += abs(current - requests[j]);
        current = requests[j];
    }

    printf("\nTotal Seek Time: %d\n", total_seek);
    printf("Average Seek Time: %.2f\n", (float)total_seek / n);
}

int main() {
    int n, head, disk_size;

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

    int requests[n];
    printf("Enter the disk requests (track numbers):\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &requests[i]);

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

    printf("Enter total number of cylinders (disk size): ");
    scanf("%d", &disk_size); // For SCAN (to simulate end of disk)

    // Make copies of request array for each algorithm
    int requests_sstf[n], requests_scan[n], requests_clook[n];
    for (int i = 0; i < n; i++) {
        requests_sstf[i] = requests[i];
        requests_scan[i] = requests[i];
        requests_clook[i] = requests[i];
```

```
    }

    SSTF(requests_sstf, n, head);
    SCAN(requests_scan, n, head, disk_size);
    CLOOK(requests_clook, n, head);

    return 0;
}
```

Output:-