**ASSIGNMENT 10**

#include <stdio.h>

#include <stdlib.h>

#define SEEK\_TIME\_PER\_TRACK 0.2 // in milliseconds

#define MAX\_TRACK 99

// Sort in ascending order

int compare(const void\* a, const void\* b) {

return (\*(int\*)a - \*(int\*)b);

}

double calculate\_seek\_time(int requests[], int n, int head, int direction) {

qsort(requests, n, sizeof(int), compare);

double total\_seek = 0.0;

int curr = head;

printf("Seek Sequence:\n");

// Step 1: move in the given direction first (up = 1)

if (direction == 1) {

// Upward

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

if (requests[i] >= head) {

printf("%d → %d\n", curr, requests[i]);

total\_seek += abs(curr - requests[i]);

curr = requests[i];

}

}

// Then reverse direction

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

if (requests[i] < head) {

printf("%d → %d\n", curr, requests[i]);

total\_seek += abs(curr - requests[i]);

curr = requests[i];

}

}

} else {

// Downward

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

if (requests[i] <= head) {

printf("%d → %d\n", curr, requests[i]);

total\_seek += abs(curr - requests[i]);

curr = requests[i];

}

}

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

if (requests[i] > head) {

printf("%d → %d\n", curr, requests[i]);

total\_seek += abs(curr - requests[i]);

curr = requests[i];

}

}

}

return total\_seek \* SEEK\_TIME\_PER\_TRACK;

}

int main() {

// Part (a)

int requests\_a[] = {32, 7, 45, 5, 10};

int num\_a = sizeof(requests\_a) / sizeof(requests\_a[0]);

int initial\_head\_a = 25;

printf("\n=== Part (a) ===\n");

double seek\_time\_a = calculate\_seek\_time(requests\_a, num\_a, initial\_head\_a, 1);

printf("Total Seek Time: %.2f ms\n", seek\_time\_a);

// Part (b)

printf("\n=== Part (b) ===\n");

int initial\_head\_b = 0;

int worst\_case\_tracks[] = {99}; // worst-case: all requests at far end

double seek\_time\_b = abs(initial\_head\_b - 99) \* SEEK\_TIME\_PER\_TRACK;

printf("Seek Sequence: 0 → 99\n");

printf("Worst-case Seek Time: %.2f ms\n", seek\_time\_b);

return 0;

}