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

#define DISKNUM 10000
#define CYLINDERNUM 1000

int cylinders[CYLINDERNUM];

int comp (const void* a, const void* b){ // Comparing Integers For Sorting
    int *x = (int*) a;
    int *y = (int*) b;
    return *x - *y;
}

void cylindersRequest(){ // Obtain Random Cylinder Requests
    srand(time(NULL)); // Different Seed For Random Number Generator
    for (int i = 0; i < CYLINDERNUM; i++){ // 1000 Cylinders Request
        while(1){ // Loop To Make Sure Value Is Unique Before Moving On
            bool found = false; // Assume Found Is False
            int random = rand() % DISKNUM; // Random Number From 0 to 9999
            for (int j = i -1 ; j > -1; j--){ // Check To See If It was Entered Already
```

```
                if(cylinders[j] == random){ // If It's Found, It's Not Unique, Break And
                    found = true; // If Found, Found = True and Reloop To Get New Random Value
                    break; // Break Out
                }
            }
            if(found == false){ // Not Fount After Looping Through Array
                //printf("%d- random: %d \n", i, random);
                cylinders[i] = random; // Requesting From a Range From 0-9999
                break; // Unique Is True So i++. Break.
            }
        }
    }
}
```

```
int FCFS(int cylinders[],int head){ // First Come First Serve Algorithm
    int sum = 0; // Sum is Zero
    for(int i = 0; i < CYLINDERNUM; i++){ // Loop From 1st To End
        sum += abs(head - cylinders[i]); // Get The Distance Traveled
        head = cylinders[i]; // New Head Value
    }
    return sum; // Return Total Head Movement
```

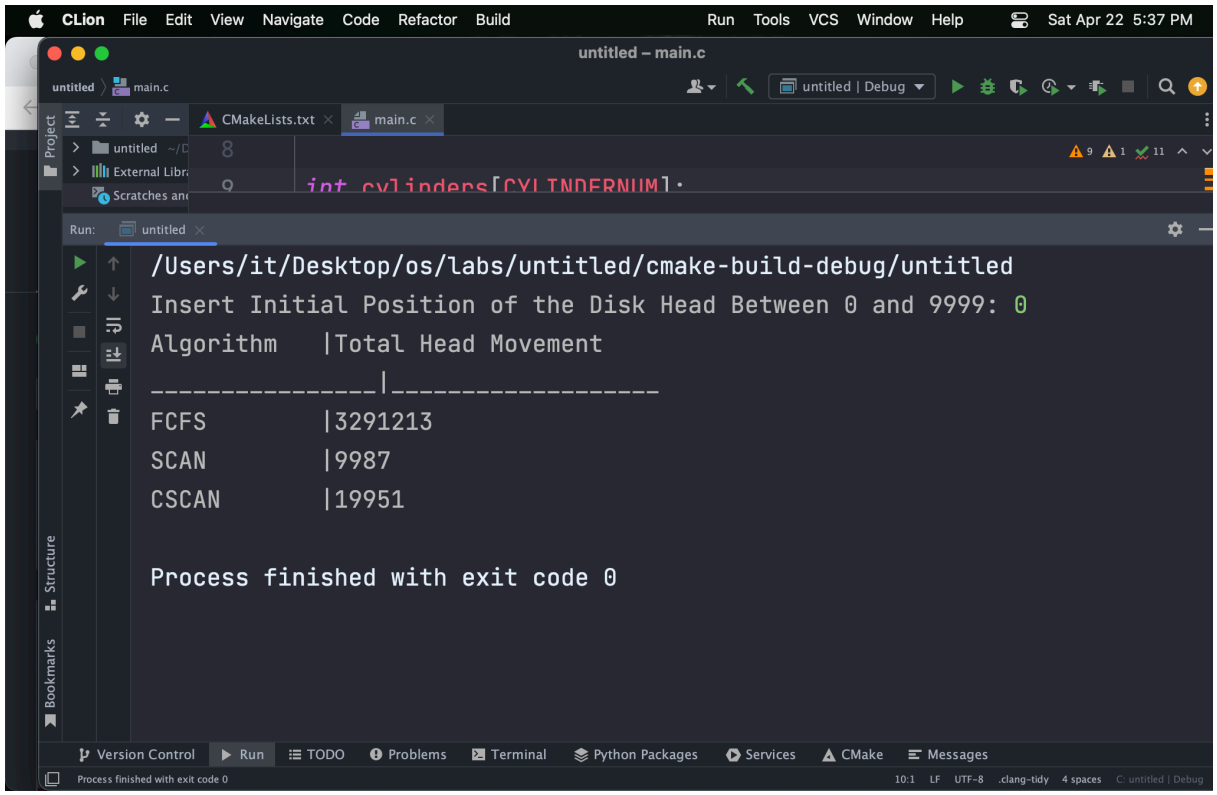
```
}

int SCAN(int cylinders[],int head){ // SCAN Algorithm
    int sum = 0; // Sum Is Zero
    int low = 0; // Amount of Values Less Than Head
    int storeLow[CYLINDERNUM]; // Used To Store Values Less Than
Current Head
    int high = 0; // Amount of Values Higher Than
    int storeHigh[CYLINDERNUM]; // Used To Store Values Higher Than
Current Head
    for(int i = 0; i < CYLINDERNUM; i++){  // Parse Array
        if(cylinders[i] > head){ // Above Head Goes Here
            storeHigh[high] = cylinders[i]; // Parse
            high++; // Increase Size Of High
        }
        else{ // Below Or Equal To Head Goes Here
            storeLow[low] = cylinders[i]; // Parse
            low++; // Increase Size Of Low
        }
    }
    qsort(storeLow, low, sizeof(*storeLow), comp); // Sort Low
    qsort(storeHigh, high, sizeof(*storeHigh), comp); // Sort High
    for(int i = low-1; i > -1; i--){ // Start To Descend From Head
        //printf("head: %d \n", head);
        sum += head - storeLow[i]; // Head Movement
        head = storeLow[i]; // New Head Value
```

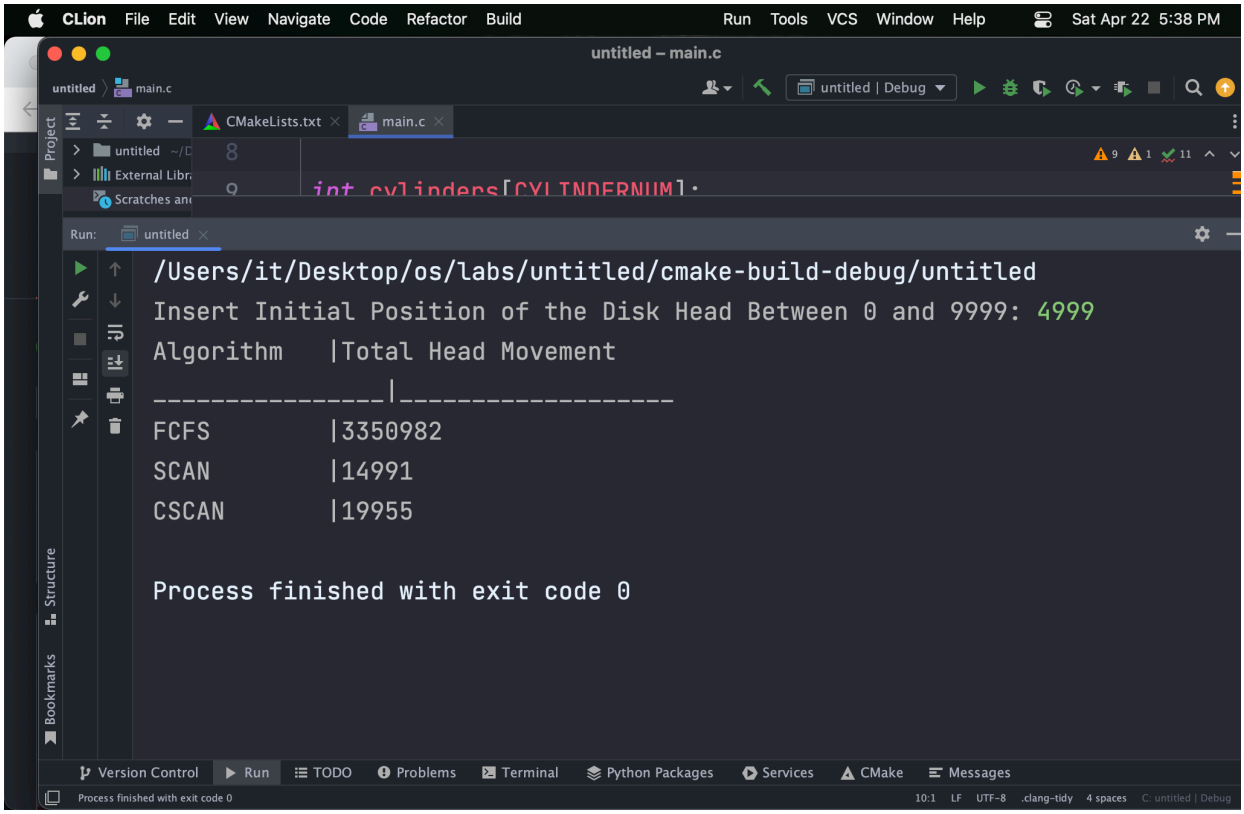
```
    }
    sum += head - 0; // To Zero
    head = 0; // Head = 0
    for(int i = 0; i < high; i++){ // Start To Ascend From Head
        //printf("head: %d \n", head);
        sum += storeHigh[i] - head; // Head Movement
        head = storeHigh[i]; // New Head Value
    }
    //printf("head: %d \n", head);
    return sum; // Return Total Head Movement
}

int CSCAN(int cylinders[],int head){ // CSCAN Algorithm
    int sum = 0; // Sum Is Zero
    int low = 0; // Amount of Values Less Than Head
    int storeLow[CYLINDERNUM]; // Used To Store Values Less Than
Current Head
    int high = 0; // Amount of Values Higher Than
    int storeHigh[CYLINDERNUM]; // Used To Store Values Higher Than
Current Head
    for(int i = 0; i < CYLINDERNUM; i++){  // Parse Array
        if(cylinders[i] > head){ // Above Head Goes Here
            storeHigh[high] = cylinders[i]; // Parse
            high++; // Increase Size Of High
        }
        else{ // Below Or Equal To Head Goes Here
```

```
            storeLow[low] = cylinders[i]; // Parse
            low++; // Increase Size Of Low
        }
    }
    qsort(storeLow, low, sizeof(*storeLow), comp); // Sort Low
    qsort(storeHigh, high, sizeof(*storeHigh), comp); // Sort High
    for(int i = low-1; i > -1; i--){ // Start To Descend From Head
        //printf("head: %d \n", head);
        sum += head - storeLow[i]; // Head Movement
        head = storeLow[i]; // New Head Value
    }
    sum += head - 0; // To Zero
    head = 0; // Head = 0
    for(int i = high-1; i > -1; i--){ // Start To Descend From Head
        //printf("head: %d \n", head);
        if (i == high-1){
            sum += storeHigh[i] - head; // Head Movement
        }
        else{
            sum += head - storeHigh[i]; // Head Movement
        }
        head = storeHigh[i]; // New Head Value
    }
    //printf("head: %d \n", head);
    return sum; // Return Total Head Movement
}
```

```
int main(){
    cylindersRequest(); // Obtain a Random Series of 1000 Cylinders Requests
    int input; // To Store Disk Head
    printf("Insert Initial Position of the Disk Head Between 0 and %d: ", DISKNUM - 1); // Print
    scanf("%d", &input); // Obtain Initial Position of the Disk Head
    while(input < 0 || input > DISKNUM - 1){ // Makes Sure Input Is Appropriate
        printf("Incorrect Input. Input Has to be Between 0 and %d. You Inputted %d.\n", DISKNUM - 1, input); // Print
        printf("Insert Initial Position of the Disk Head Between 0 and %d: ", DISKNUM -1); // Print
        scanf("%d", &input); // Obtain Initial Position Of The Disk Head
    }
    int FCFS_SUM = FCFS(cylinders,input); // Store
    int SCAN_SUM = SCAN(cylinders,input); // Store
    int CSCAN_SUM = CSCAN(cylinders,input); // Store

    printf("Algorithm\t|Total Head Movement\n"); // Print
    printf("_______________|___________________\n"); // Print
    printf("FCFS\t\t|%d\n", FCFS_SUM); // Print
```

```c
    printf("SCAN\t\t|%d\n", SCAN_SUM); // Print
    printf("CSCAN\t\t|%d\n", CSCAN_SUM); // Print
    return 0;
}
```





```
/Users/it/Desktop/os/labs/untitled/cmake-build-debug/untitled
Insert Initial Position of the Disk Head Between 0 and 9999: 9999
Algorithm    |Total Head Movement
_____________|___________________
FCFS         |3380216
SCAN         |9999
CSCAN        |9999

Process finished with exit code 0
```

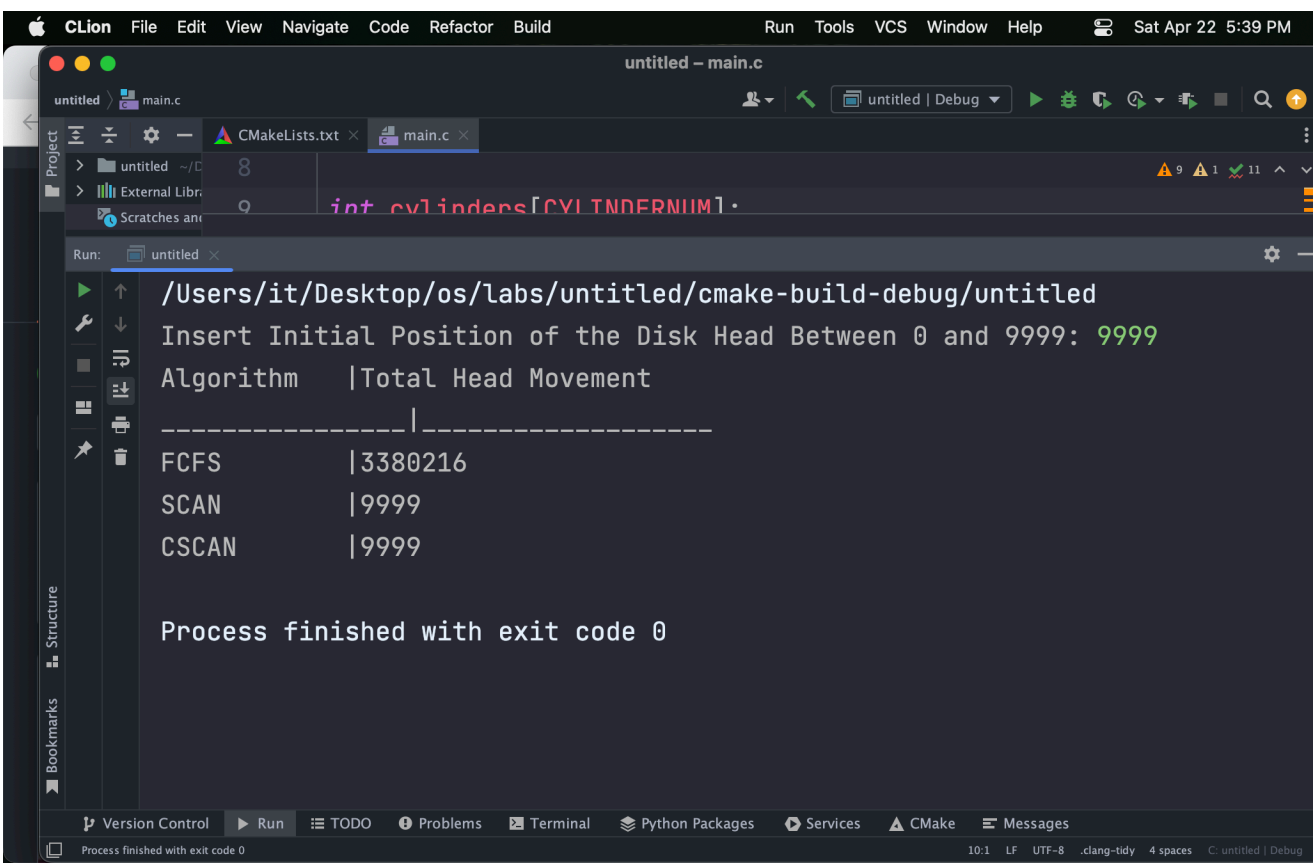
```
/Users/it/Desktop/os/labs/untitled/cmake-build-debug/untitled
Insert Initial Position of the Disk Head Between 0 and 9999: 10000
Incorrect Input. Input Has to be Between 0 and 9999. You Inputted 10000.
Insert Initial Position of the Disk Head Between 0 and 9999:
```

1. **FCFS** algorithm abbreviation for First Come First Serve, as a name imply like grocery line, whoever come first to the counter, that person will be checkout first. Therefore it is straightforward and simple, but it is not the fastest. This algorithm receive requests in the order they arrive in the disk queue.

SCAN (elevator) algorithm using elevator principle. If you are on 3rd floor and you need to go to first floor but other people before you already requested to go up (7, 9 floors). So elevator first go up to fulfill their request and once fulfill, go to your desired floor. Head start from one end of the disk and moves towards the other end of disk and servicing requests one by one and reach the other end and turn around and goes to other end. This algorithm is better then FCFS.

CSCAN algorithm is improved version of elevator algorithm. Once it is reach one end it goes back to the beginning without taking any requests.

2.



3. It seems to me that SCAN and CSCAN is pretty much the same, each one has own advantage and disadvantage. For example SCAN takes longer waiting time than the C-SCAN scheduling algorithm for requesting the locations and C-SCAN algorithm provides uniform waiting time and better response time. Under a light load, SCAN policy is best. But under a heavy load, C-SCAN policy is best. In SCAN the head moves till the end of the disk despite the absence of requests to be serviced. More seek movements are caused in C-SCAN compared to SCAN Algorithm. They both use quick sort algorithm.
And FCFS is least efficient among all of them because it have to loop though all data in order they arrived.