31. Construct a C program to simulate the First in First Out paging technique of memory management.

#include <stdio.h>

int main() {

int n, capacity, i, j, page, faults = 0, flag, pos = 0;

printf("Enter number of pages: ");

scanf("%d", &n);

int pages[n];

printf("Enter the reference string (pages): ");

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

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

printf("Enter capacity of frames: ");

scanf("%d", &capacity);

int frames[capacity];

for (i = 0; i < capacity; i++)

frames[i] = -1; // Initialize frames as empty

printf("\nPage Reference\tFrames\n");

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

page = pages[i];

flag = 0;

// Check if page is already in frames (page hit)

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

if (frames[j] == page) {

flag = 1;

break;

}

}

if (flag == 0) { // Page fault

frames[pos] = page; // Replace frame using FIFO position

pos = (pos + 1) % capacity;

faults++;

}

// Print current page and frame status

printf("%d\t\t", page);

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

if (frames[j] != -1)

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

else

printf("- ");

}

printf("\n");

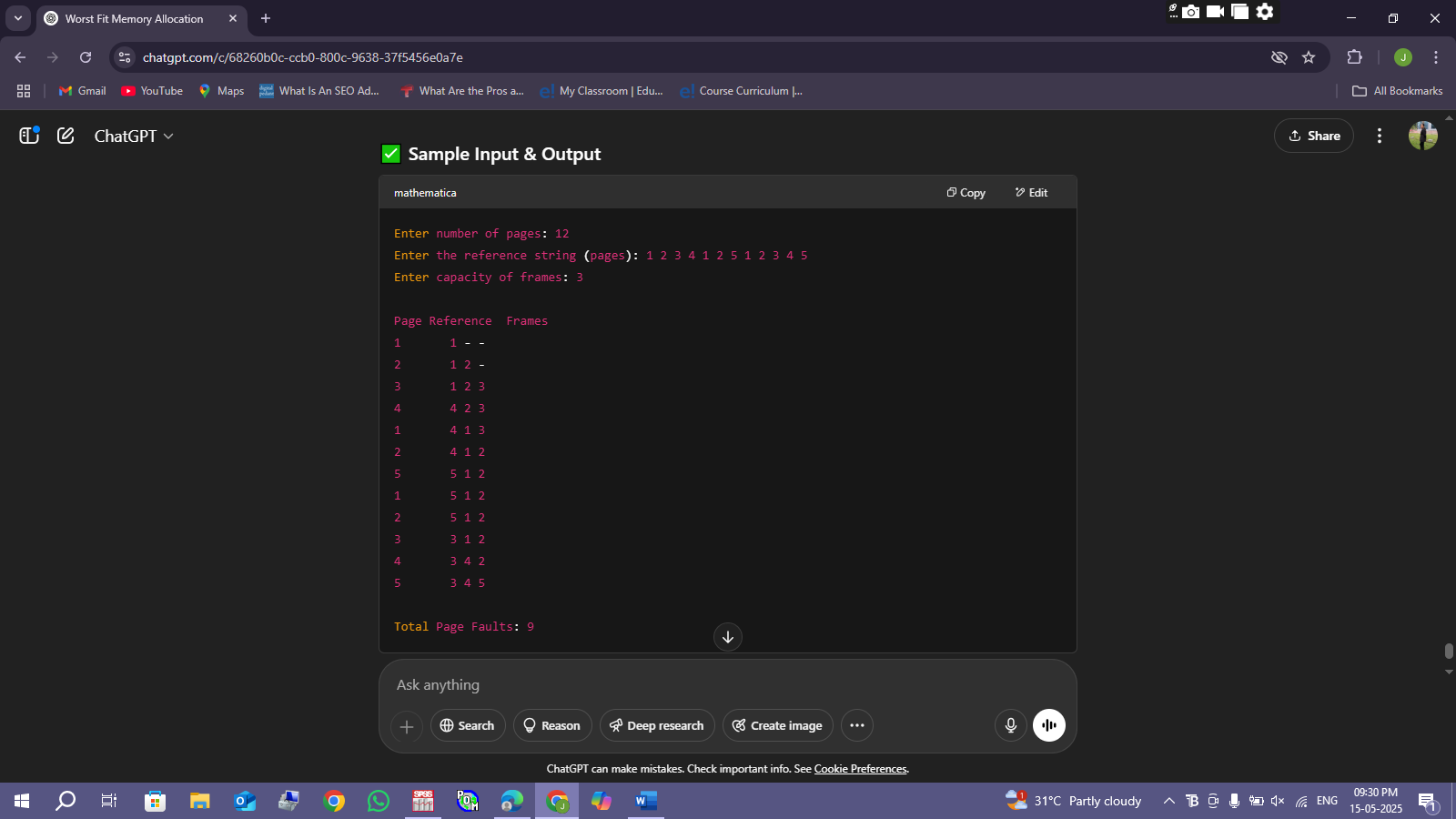
}

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

return 0;

}

OUTPUT:



32. Construct a C program to simulate the Least Recently Used paging technique of memory management.

#include <stdio.h>

int main() {

int n, capacity, i, j, page, faults = 0;

printf("Enter number of pages: ");

scanf("%d", &n);

int pages[n];

printf("Enter the reference string (pages): ");

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

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

printf("Enter capacity of frames: ");

scanf("%d", &capacity);

int frames[capacity];

int recent[capacity]; // To store recent usage index for each frame

// Initialize frames and recent arrays

for (i = 0; i < capacity; i++) {

frames[i] = -1;

recent[i] = -1;

}

printf("\nPage Reference\tFrames\n");

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

page = pages[i];

int flag = 0;

// Check if page is already in frame (hit)

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

if (frames[j] == page) {

flag = 1;

recent[j] = i; // Update recent usage

break;

}

}

if (flag == 0) {

// Page fault: find least recently used frame

int lru\_index = 0;

int min\_recent = recent[0];

for (j = 1; j < capacity; j++) {

if (recent[j] < min\_recent) {

min\_recent = recent[j];

lru\_index = j;

}

}

frames[lru\_index] = page;

recent[lru\_index] = i;

faults++;

}

// Print page and frames content

printf("%d\t\t", page);

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

if (frames[j] != -1)

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

else

printf("- ");

}

printf("\n");

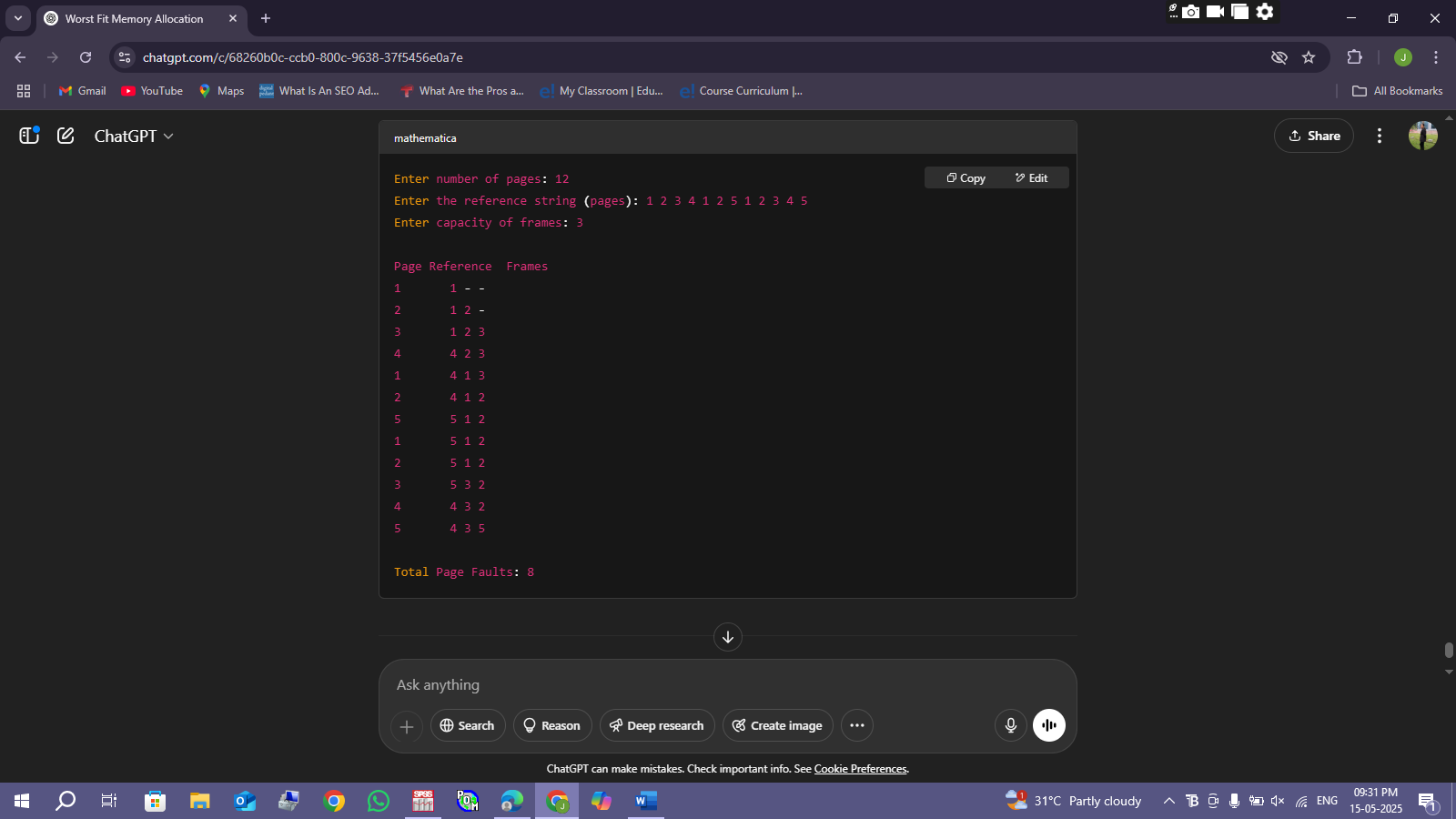
}

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

return 0;

}

OUTPUT:



33.Construct a C program to simulate the optimal paging technique of memory management

#include <stdio.h>

int findOptimal(int pages[], int frames[], int n, int capacity, int start) {

int i, j;

int res = -1, farthest = start;

for (i = 0; i < capacity; i++) {

int found = 0;

for (j = start; j < n; j++) {

if (frames[i] == pages[j]) {

if (j > farthest) {

farthest = j;

res = i;

}

found = 1;

break;

}

}

if (!found) {

// If the page is never referenced again

return i;

}

}

if (res == -1) return 0; // If all pages are referenced immediately, replace first frame

else return res;

}

int main() {

int n, capacity, i, j, page, faults = 0;

printf("Enter number of pages: ");

scanf("%d", &n);

int pages[n];

printf("Enter the reference string (pages): ");

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

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

printf("Enter capacity of frames: ");

scanf("%d", &capacity);

int frames[capacity];

for (i = 0; i < capacity; i++)

frames[i] = -1; // Initialize frames empty

printf("\nPage Reference\tFrames\n");

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

page = pages[i];

int flag = 0;

// Check if page already in frames (hit)

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

if (frames[j] == page) {

flag = 1;

break;

}

}

if (flag == 0) { // Page fault

faults++;

// Find frame to replace using optimal algorithm

int replace\_index = findOptimal(pages, frames, n, capacity, i + 1);

frames[replace\_index] = page;

}

// Print current page and frames

printf("%d\t\t", page);

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

if (frames[j] != -1)

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

else

printf("- ");

}

printf("\n");

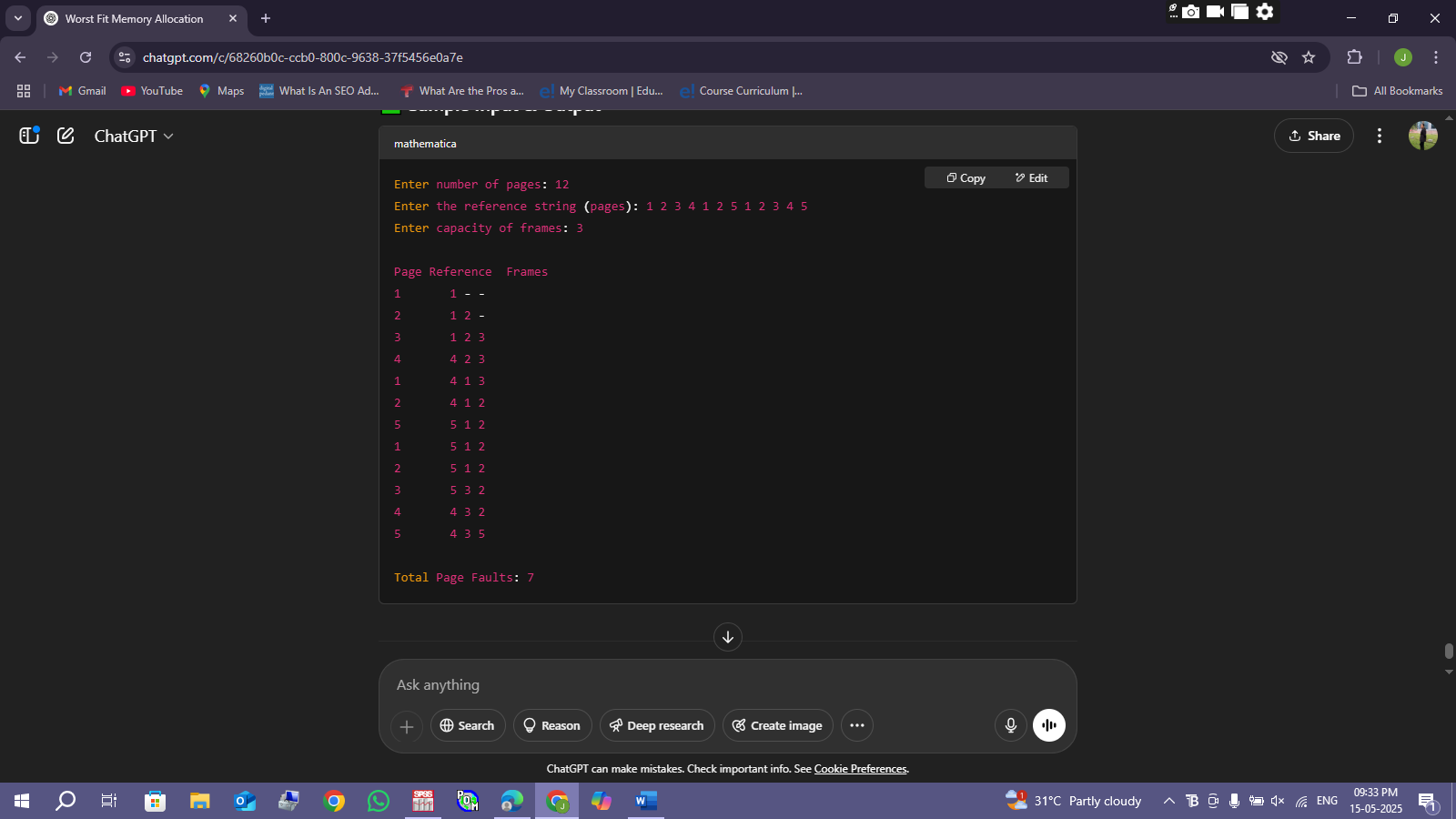
}

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

return 0;

}

OUTPUT:



34. Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a C program to simulate the file allocation strategy.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_RECORDS 100

#define MAX\_RECORD\_SIZE 100

char fileSystem[MAX\_RECORDS][MAX\_RECORD\_SIZE];

int recordCount = 0;

// Function to add a record sequentially

void addRecord(char record[]) {

if (recordCount >= MAX\_RECORDS) {

printf("File is full. Cannot add more records.\n");

return;

}

strcpy(fileSystem[recordCount], record);

recordCount++;

printf("Record added successfully.\n");

}

// Function to read a specific record (requires reading all previous)

void readRecord(int recNum) {

if (recNum < 1 || recNum > recordCount) {

printf("Invalid record number.\n");

return;

}

printf("Reading records sequentially to reach record %d:\n", recNum);

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

printf("Record %d: %s\n", i + 1, fileSystem[i]);

}

}

int main() {

int choice;

char record[MAX\_RECORD\_SIZE];

int recNum;

printf("Sequential File Allocation Simulation\n");

while (1) {

printf("\n1. Add Record\n2. Read Record\n3. Exit\nEnter your choice: ");

scanf("%d", &choice);

getchar(); // consume newline after number input

switch (choice) {

case 1:

printf("Enter record data: ");

fgets(record, MAX\_RECORD\_SIZE, stdin);

// Remove trailing newline

record[strcspn(record, "\n")] = '\0';

addRecord(record);

break;

case 2:

printf("Enter record number to read: ");

scanf("%d", &recNum);

readRecord(recNum);

break;

case 3:

printf("Exiting simulation.\n");

exit(0);

default:

printf("Invalid choice! Try again.\n");

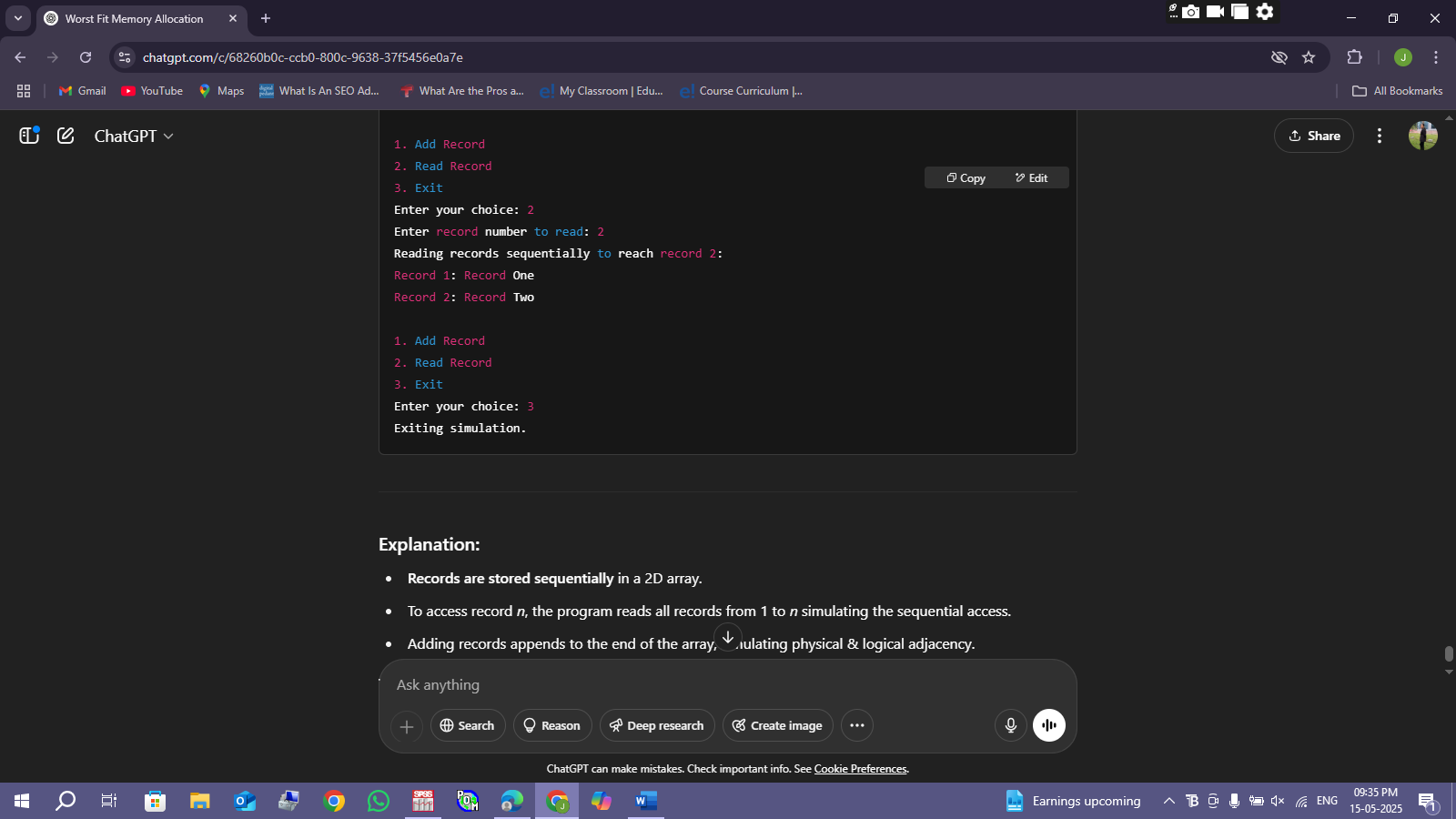
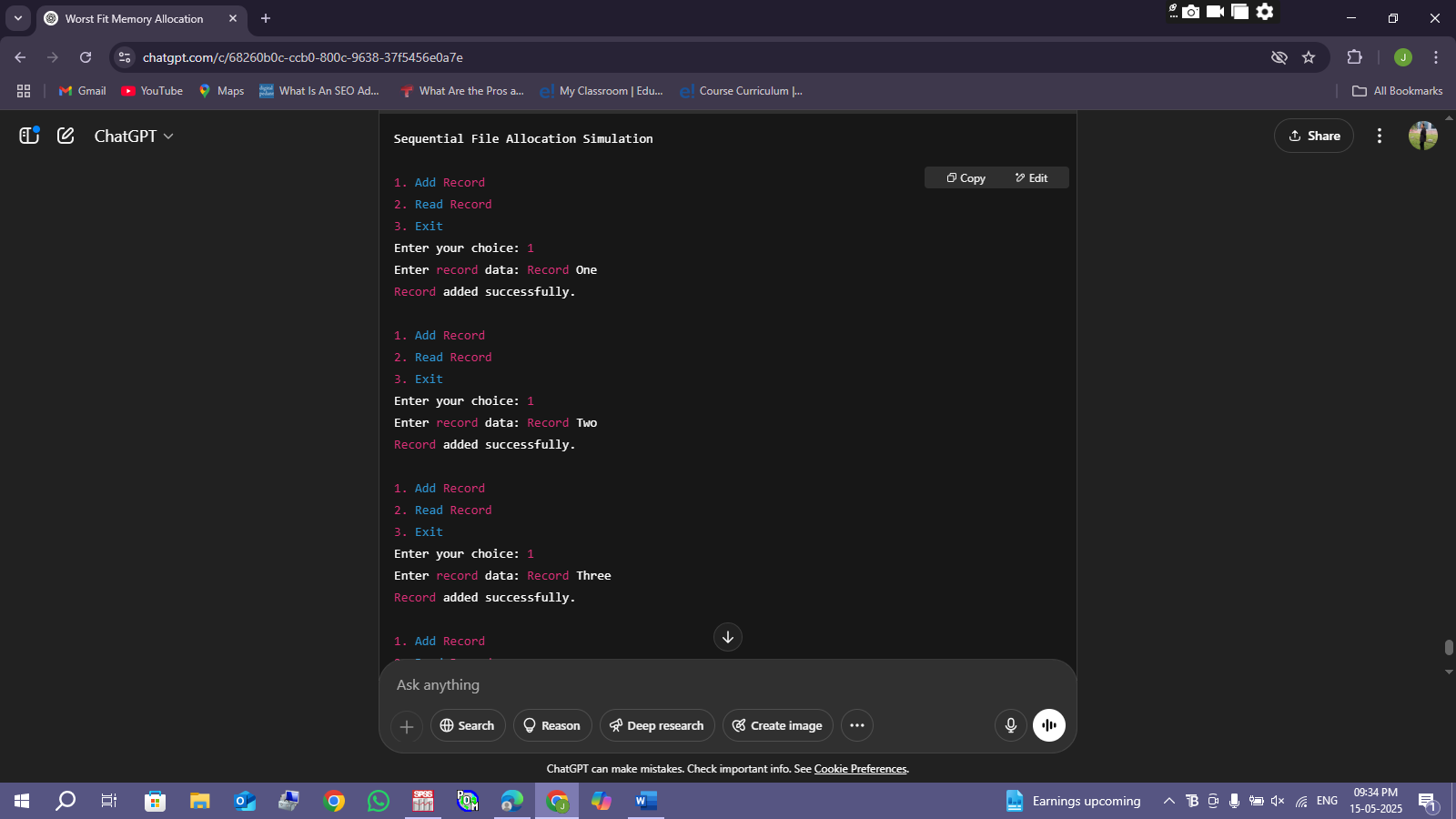
}

}

return 0;

}

OUTPUT:



35. Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a C program to simulate the file allocation strategy.

#include <stdio.h>

#include <stdlib.h>

#define MAX\_BLOCKS 100

#define BLOCK\_SIZE 512 // Just a conceptual block size

// Structure to represent a file

typedef struct {

int indexBlock[MAX\_BLOCKS]; // Index block storing pointers to data blocks

int totalBlocks; // Number of blocks allocated to the file

} File;

void initializeFile(File\* file) {

file->totalBlocks = 0;

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

file->indexBlock[i] = -1; // -1 means no block allocated

}

}

void allocateBlock(File\* file, int blockNumber) {

if (file->totalBlocks >= MAX\_BLOCKS) {

printf("No more blocks can be allocated.\n");

return;

}

file->indexBlock[file->totalBlocks] = blockNumber;

file->totalBlocks++;

printf("Block %d allocated at index %d in index block.\n", blockNumber, file->totalBlocks - 1);

}

void displayFileBlocks(File\* file) {

if (file->totalBlocks == 0) {

printf("No blocks allocated to the file.\n");

return;

}

printf("File blocks pointed by index block:\n");

for (int i = 0; i < file->totalBlocks; i++) {

printf("Index %d -> Block %d\n", i, file->indexBlock[i]);

}

}

int main() {

File file;

int choice, blockNum;

initializeFile(&file);

printf("Indexed File Allocation Simulation\n");

while (1) {

printf("\n1. Allocate Block\n2. Display File Blocks\n3. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter block number to allocate: ");

scanf("%d", &blockNum);

allocateBlock(&file, blockNum);

break;

case 2:

displayFileBlocks(&file);

break;

case 3:

printf("Exiting simulation.\n");

exit(0);

default:

printf("Invalid choice! Try again.\n");

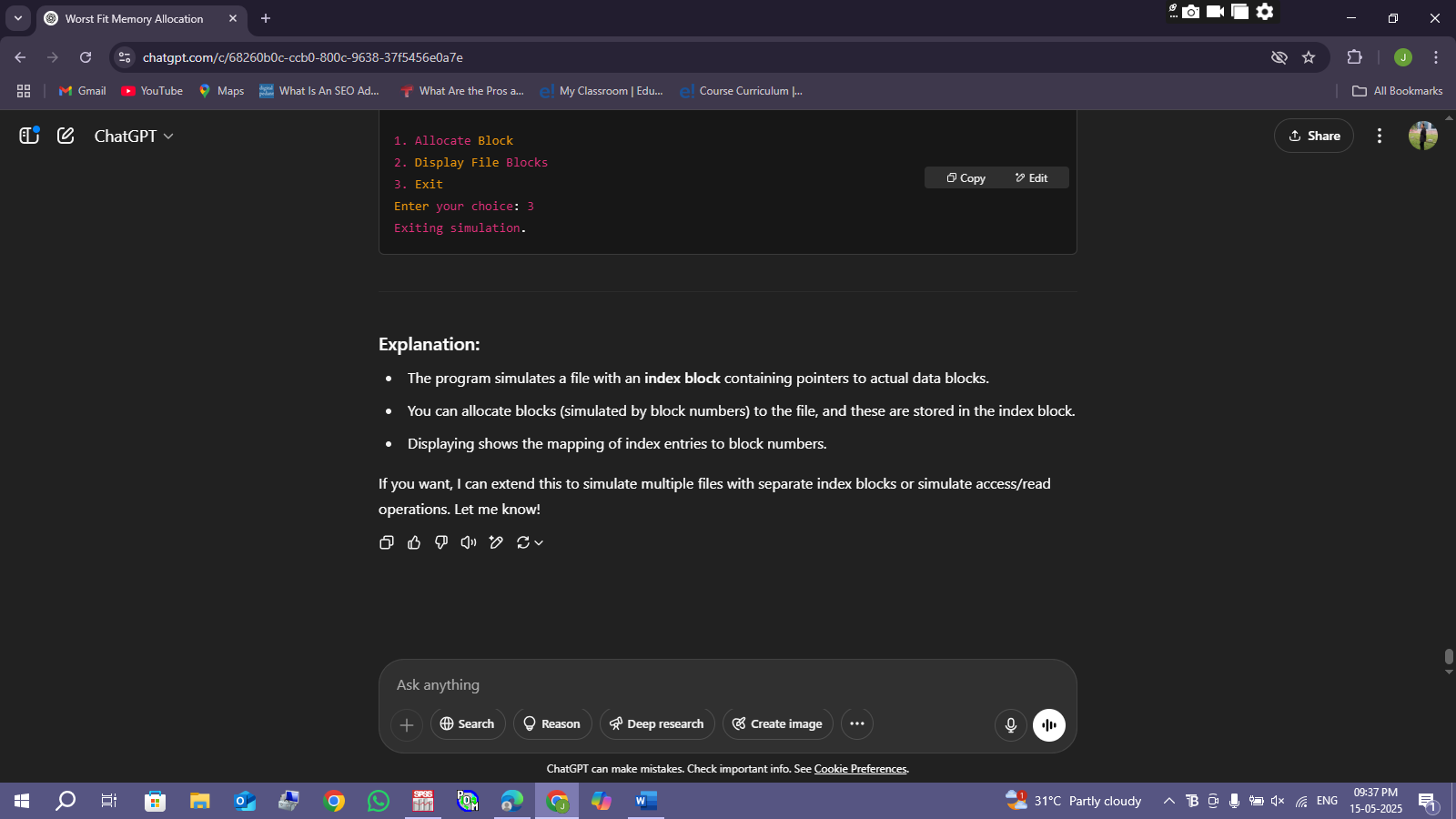
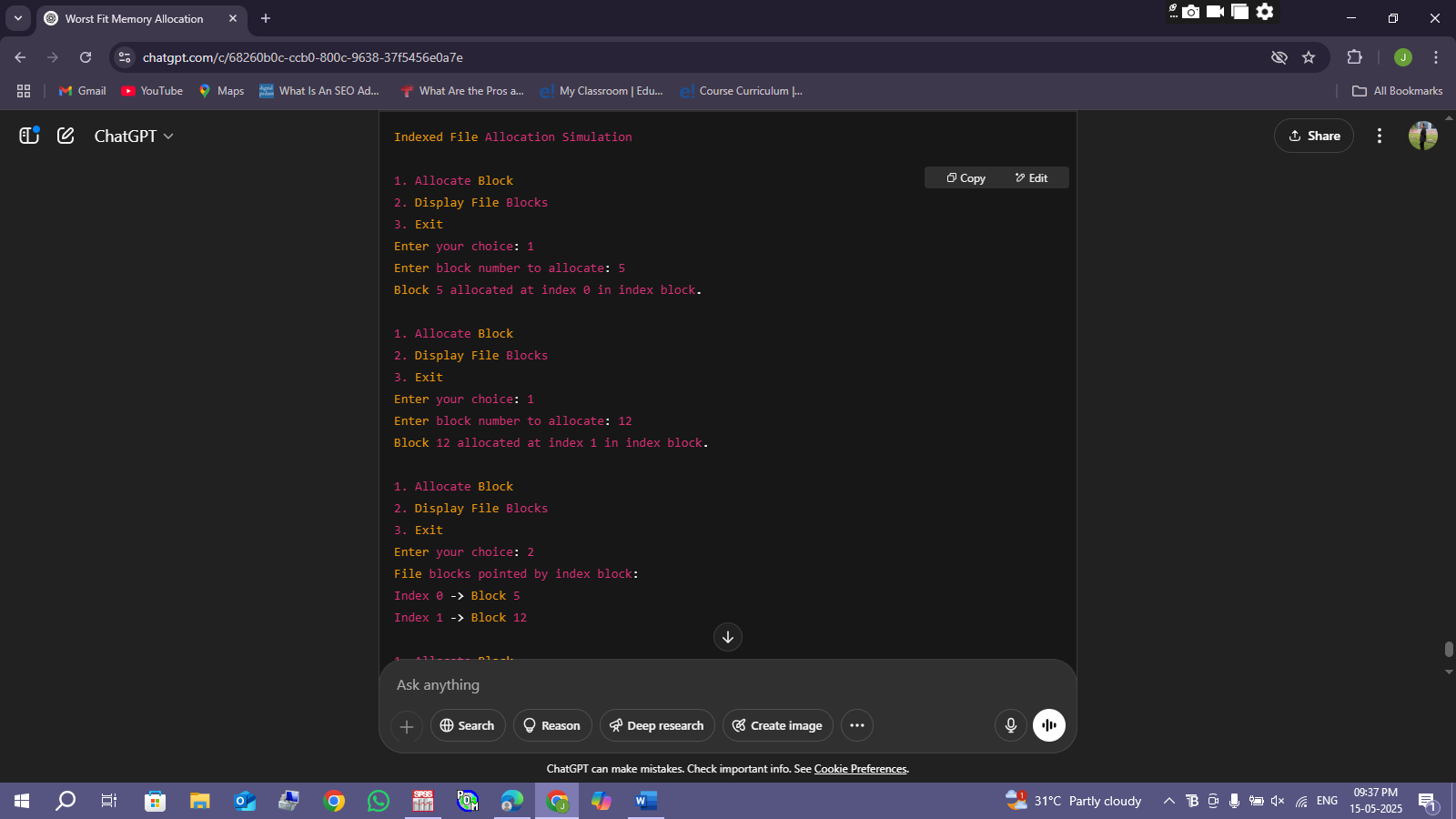
}

}

return 0;

}

OUTPUT:



36.With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a C program to simulate the file allocation strategy.

#include <stdio.h>

#include <stdlib.h>

typedef struct Block {

int blockNumber;

struct Block\* next;

} Block;

typedef struct {

Block\* head; // Pointer to first block

Block\* tail; // Pointer to last block

} File;

void initializeFile(File\* file) {

file->head = NULL;

file->tail = NULL;

}

void allocateBlock(File\* file, int blockNum) {

Block\* newBlock = (Block\*)malloc(sizeof(Block));

if (!newBlock) {

printf("Memory allocation failed.\n");

return;

}

newBlock->blockNumber = blockNum;

newBlock->next = NULL;

if (file->head == NULL) {

// First block of the file

file->head = newBlock;

file->tail = newBlock;

} else {

// Append at the end

file->tail->next = newBlock;

file->tail = newBlock;

}

printf("Block %d allocated and linked.\n", blockNum);

}

void displayFileBlocks(File\* file) {

if (file->head == NULL) {

printf("No blocks allocated to the file.\n");

return;

}

printf("File blocks linked list:\n");

Block\* current = file->head;

while (current != NULL) {

printf("Block %d", current->blockNumber);

if (current->next != NULL)

printf(" -> ");

current = current->next;

}

printf("\n");

}

void freeFileBlocks(File\* file) {

Block\* current = file->head;

while (current != NULL) {

Block\* temp = current;

current = current->next;

free(temp);

}

file->head = NULL;

file->tail = NULL;

}

int main() {

File file;

int choice, blockNum;

initializeFile(&file);

printf("Linked File Allocation Simulation\n");

while (1) {

printf("\n1. Allocate Block\n2. Display File Blocks\n3. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter block number to allocate: ");

scanf("%d", &blockNum);

allocateBlock(&file, blockNum);

break;

case 2:

displayFileBlocks(&file);

break;

case 3:

freeFileBlocks(&file);

printf("Exiting simulation.\n");

exit(0);

default:

printf("Invalid choice! Try again.\n");

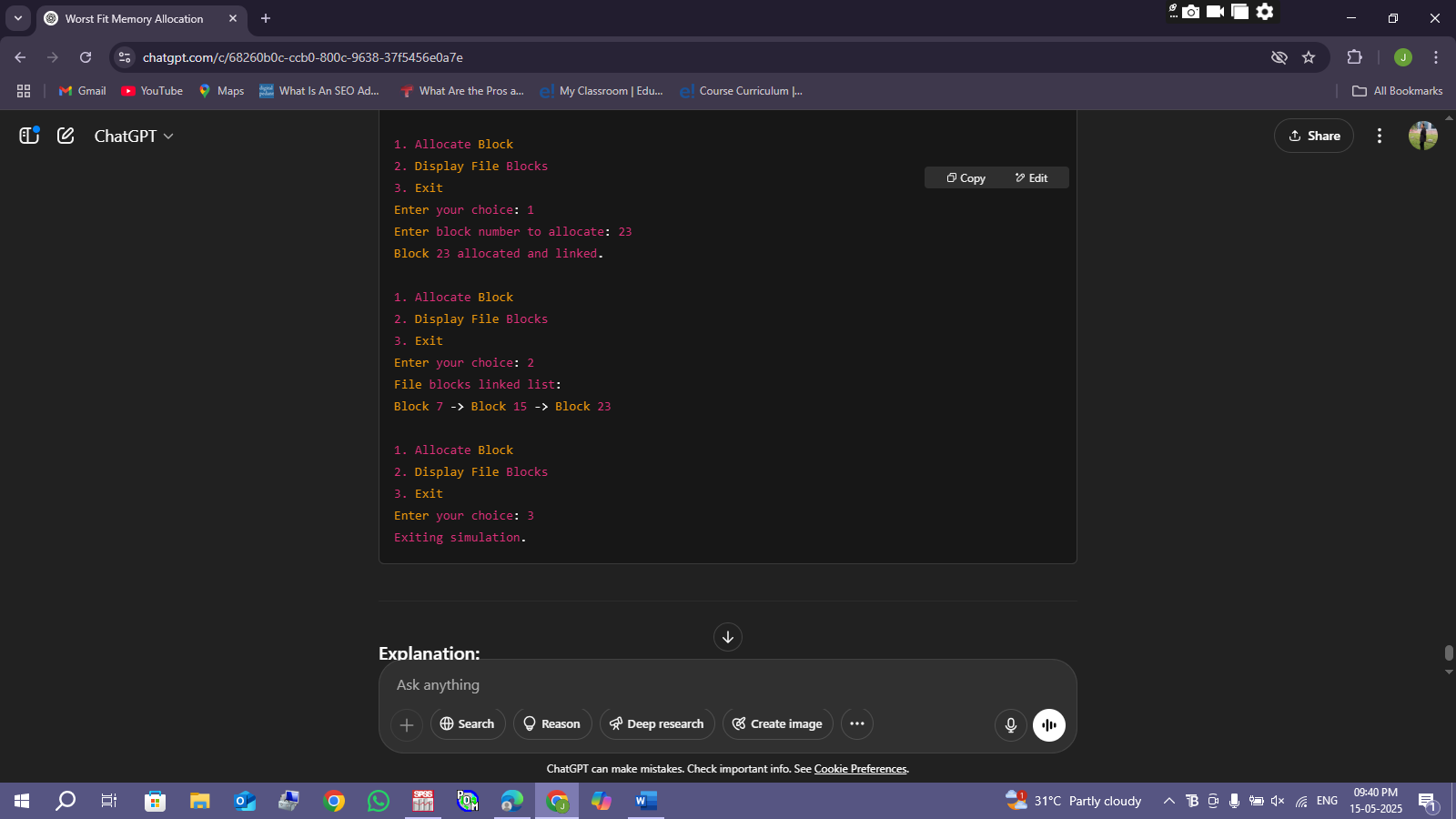
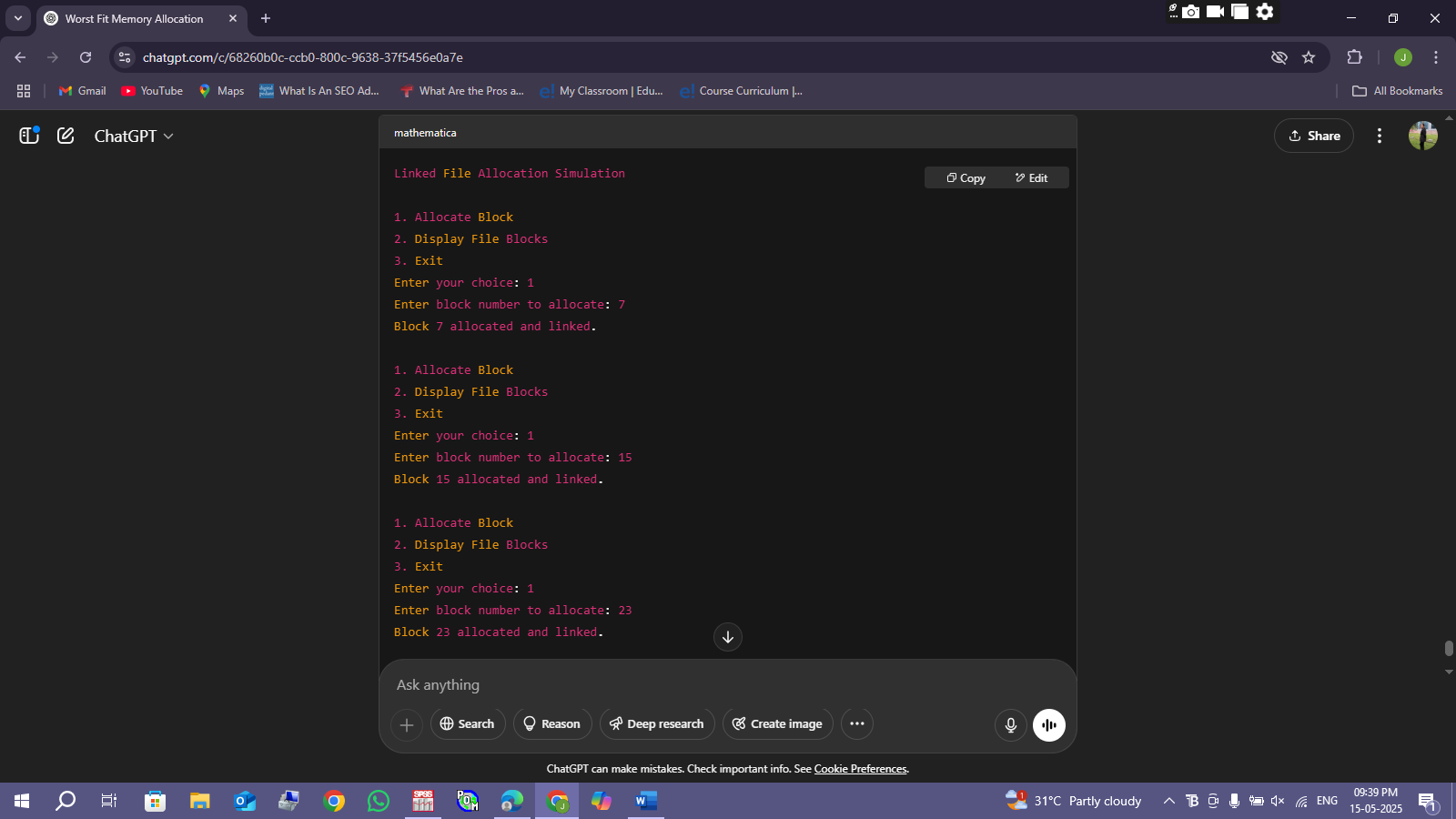
}

}

return 0;

}

OUTPUT:



37.Construct a C program to simulate the First Come First Served disk scheduling algorithm.

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int main() {

int n, i;

int head, seek\_count = 0;

int \*requests;

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

scanf("%d", &n);

requests = (int \*)malloc(n \* sizeof(int));

if (requests == NULL) {

printf("Memory allocation failed.\n");

return 1;

}

printf("Enter the disk requests sequence:\n");

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

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

}

printf("Enter initial head position: ");

scanf("%d", &head);

printf("\nServicing order:\n");

printf("%d", head);

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

int distance = abs(requests[i] - head);

seek\_count += distance;

head = requests[i];

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

}

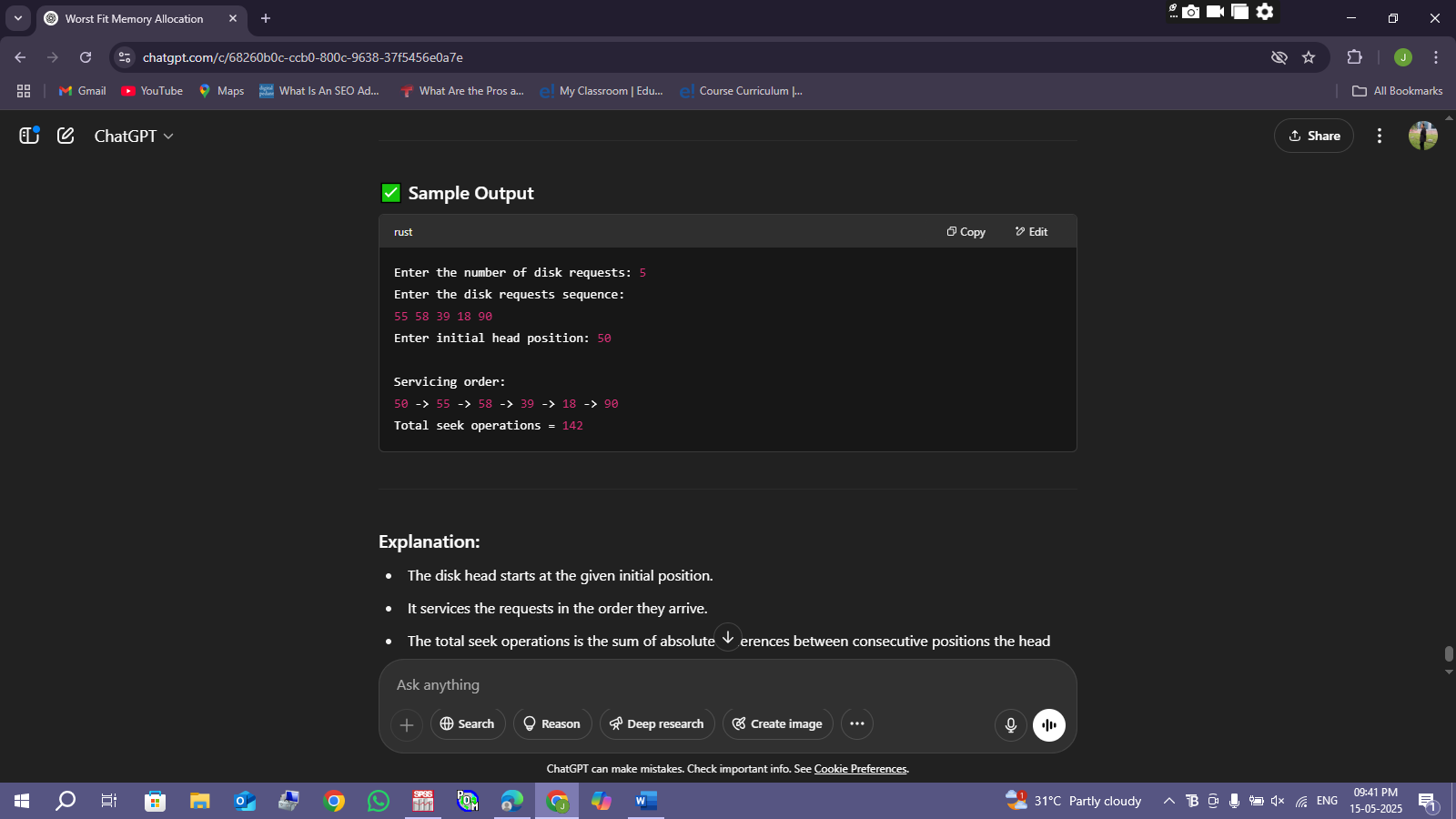
printf("\nTotal seek operations = %d\n", seek\_count);

free(requests);

return 0;

}

OUTPUT:



38. Design a C program to simulate SCAN disk scheduling algorithm.

#include <stdio.h>

#include <stdlib.h>

void scan(int requests[], int n, int head, int disk\_size, int direction) {

int seek\_count = 0;

int distance, cur\_track;

int \*left = (int \*)malloc(n \* sizeof(int));

int \*right = (int \*)malloc(n \* sizeof(int));

int l = 0, r = 0;

int i;

// Separate requests into left and right of head

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

if (requests[i] < head)

left[l++] = requests[i];

else

right[r++] = requests[i];

}

// Add the end points depending on direction

if (direction == 0) // moving left

left[l++] = 0;

else // moving right

right[r++] = disk\_size - 1;

// Sort left and right arrays

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

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

}

qsort(left, l, sizeof(int), compare);

qsort(right, r, sizeof(int), compare);

printf("Seek sequence:\n");

printf("%d ", head);

if (direction == 1) {

// Move right first

for (i = 0; i < r; i++) {

cur\_track = right[i];

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

printf("-> %d ", cur\_track);

}

// Then move left

for (i = l - 1; i >= 0; i--) {

cur\_track = left[i];

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

printf("-> %d ", cur\_track);

}

} else {

// Move left first

for (i = l - 1; i >= 0; i--) {

cur\_track = left[i];

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

printf("-> %d ", cur\_track);

}

// Then move right

for (i = 0; i < r; i++) {

cur\_track = right[i];

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

printf("-> %d ", cur\_track);

}

}

printf("\nTotal seek operations = %d\n", seek\_count);

free(left);

free(right);

}

int main() {

int n, head, disk\_size, direction, i;

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

scanf("%d", &n);

int requests[n];

printf("Enter the disk requests sequence:\n");

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

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

}

printf("Enter initial head position: ");

scanf("%d", &head);

printf("Enter disk size (number of tracks): ");

scanf("%d", &disk\_size);

printf("Enter initial head movement direction (1 for right, 0 for left): ");

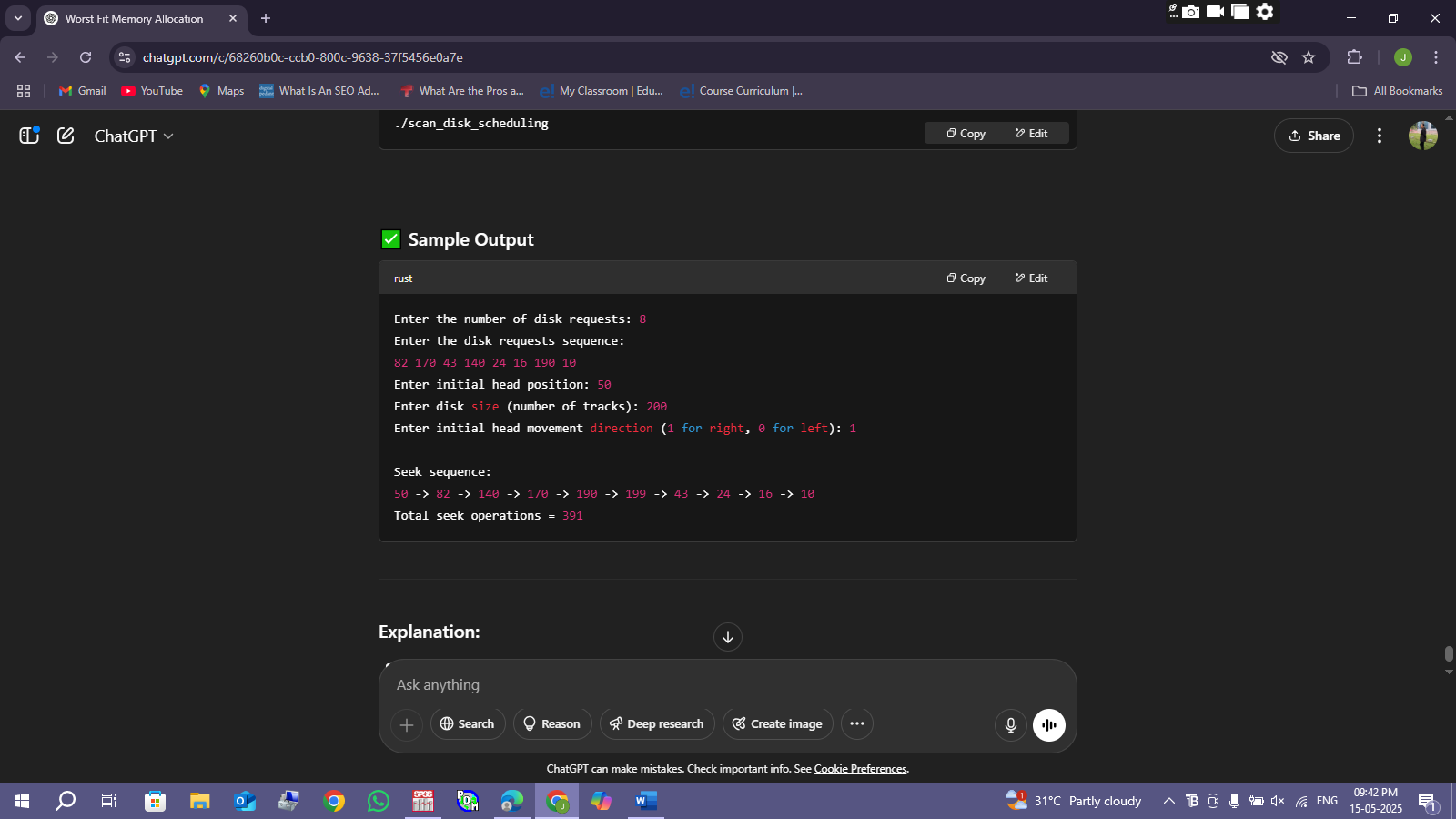
scanf("%d", &direction);

scan(requests, n, head, disk\_size, direction);

return 0;

}

OUTPUT:



39. Develop a C program to simulate C-SCAN disk scheduling algorithm.

#include <stdio.h>

#include <stdlib.h>

void cscan(int requests[], int n, int head, int disk\_size) {

int seek\_count = 0;

int distance, cur\_track;

int \*left = (int \*)malloc(n \* sizeof(int));

int \*right = (int \*)malloc(n \* sizeof(int));

int l = 0, r = 0;

int i;

// Separate requests into left and right of head

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

if (requests[i] < head)

left[l++] = requests[i];

else

right[r++] = requests[i];

}

// Add the end points to left and right

left[l++] = 0;

right[r++] = disk\_size - 1;

// Sort left and right arrays

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

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

}

qsort(left, l, sizeof(int), compare);

qsort(right, r, sizeof(int), compare);

printf("Seek sequence:\n");

printf("%d ", head);

// Move right servicing requests

for (i = 0; i < r; i++) {

cur\_track = right[i];

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

printf("-> %d ", cur\_track);

}

// Jump from end to beginning (no servicing)

distance = abs((disk\_size - 1) - 0);

seek\_count += distance;

head = 0;

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

// Service left requests

for (i = 0; i < l - 1; i++) { // exclude the 0 already serviced above

cur\_track = left[i];

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

printf("-> %d ", cur\_track);

}

printf("\nTotal seek operations = %d\n", seek\_count);

free(left);

free(right);

}

int main() {

int n, head, disk\_size, i;

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

scanf("%d", &n);

int requests[n];

printf("Enter the disk requests sequence:\n");

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

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

}

printf("Enter initial head position: ");

scanf("%d", &head);

printf("Enter disk size (number of tracks): ");

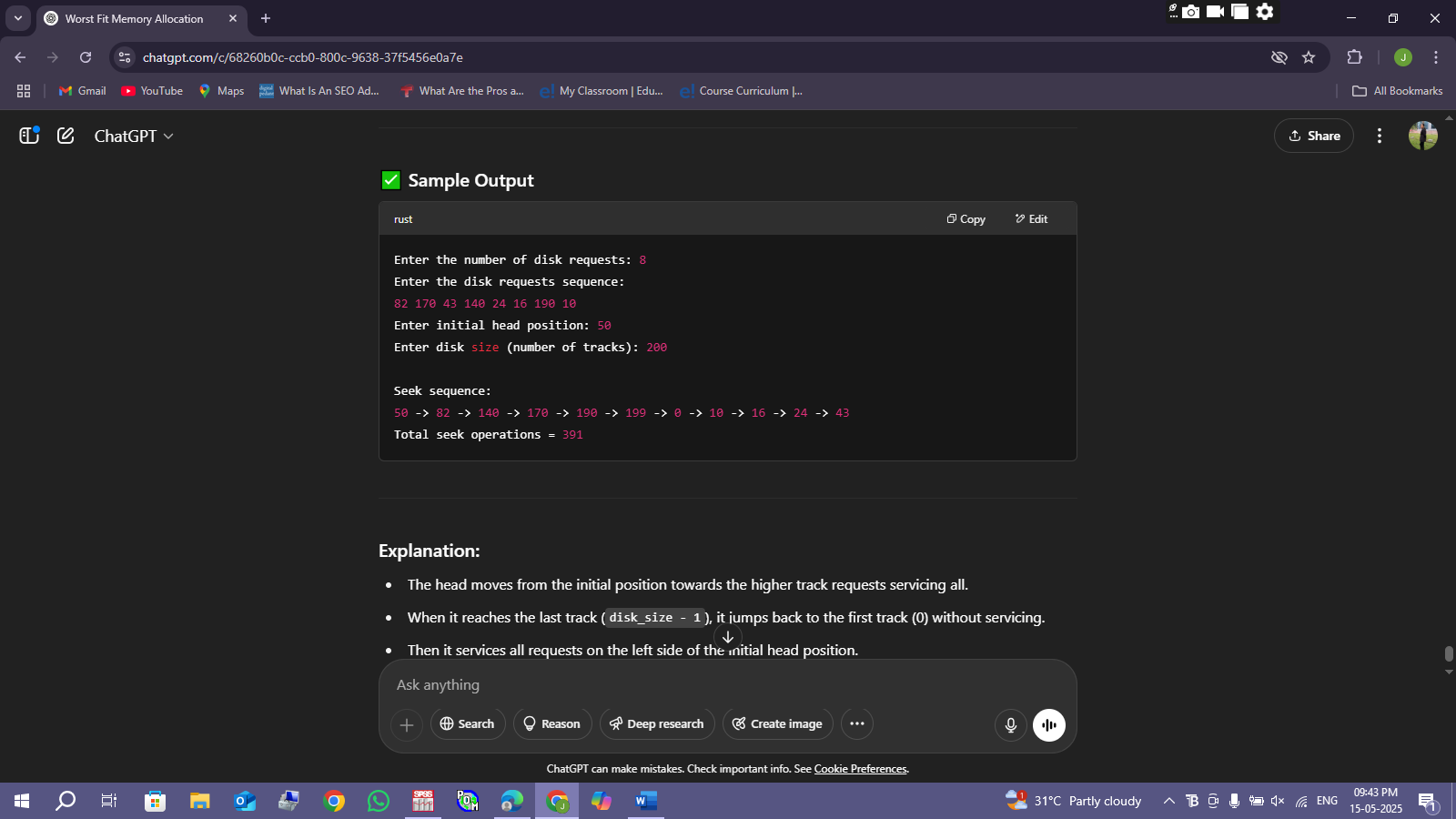
scanf("%d", &disk\_size);

cscan(requests, n, head, disk\_size);

return 0;

}

OUTPUT:



40. Illustrate the various File Access Permission and different types users in Linux.

#include <stdio.h>

#include <sys/stat.h>

#include <pwd.h>

#include <grp.h>

#include <time.h>

void printPermissions(mode\_t mode) {

printf( (mode & S\_IRUSR) ? "r" : "-");

printf( (mode & S\_IWUSR) ? "w" : "-");

printf( (mode & S\_IXUSR) ? "x" : "-");

printf( (mode & S\_IRGRP) ? "r" : "-");

printf( (mode & S\_IWGRP) ? "w" : "-");

printf( (mode & S\_IXGRP) ? "x" : "-");

printf( (mode & S\_IROTH) ? "r" : "-");

printf( (mode & S\_IWOTH) ? "w" : "-");

printf( (mode & S\_IXOTH) ? "x" : "-");

}

int main(int argc, char \*argv[]) {

if (argc != 2) {

printf("Usage: %s <filename>\n", argv[0]);

return 1;

}

struct stat fileStat;

if (stat(argv[1], &fileStat) < 0) {

perror("stat");

return 1;

}

printf("File: %s\n", argv[1]);

printf("Permissions: ");

printPermissions(fileStat.st\_mode);

printf("\n");

// Print file type

printf("File type: ");

if (S\_ISREG(fileStat.st\_mode)) printf("Regular file\n");

else if (S\_ISDIR(fileStat.st\_mode)) printf("Directory\n");

else if (S\_ISLNK(fileStat.st\_mode)) printf("Symbolic link\n");

else printf("Other\n");

// Owner info

struct passwd \*pw = getpwuid(fileStat.st\_uid);

printf("Owner: %s (UID %d)\n", pw ? pw->pw\_name : "Unknown", fileStat.st\_uid);

// Group info

struct group \*gr = getgrgid(fileStat.st\_gid);

printf("Group: %s (GID %d)\n", gr ? gr->gr\_name : "Unknown", fileStat.st\_gid);

// File size

printf("Size: %ld bytes\n", fileStat.st\_size);

// Last modified time

char timebuf[100];

struct tm \*tm\_info = localtime(&fileStat.st\_mtime);

strftime(timebuf, sizeof(timebuf), "%Y-%m-%d %H:%M:%S", tm\_info);

printf("Last modified: %s\n", timebuf);

return 0;

}

OUTPUT:

