6.Write a C program to simulate the following contiguous memory allocation techniques a)Worst-fit ,b) Best-fit, c) First-fit Can do anyone of this

#include <stdio.h>

// Define structure Block struct Block { int block\_no; int block\_size; int is\_free; };

// Define structure File struct File { int file\_no; int file\_size; };

// Function for Best Fit Allocation void bestFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) { printf("\nMemory Management Scheme - Best Fit\n"); printf("File\_no\tFile\_size\tBlock\_no\tBlock\_size\n");

for (int i = 0; i < n\_files; i++) {  
 int best\_idx = -1;  
  
 for (int j = 0; j < n\_blocks; j++) {  
 if (blocks[j].is\_free && blocks[j].block\_size >= files[i].file\_size) {  
 if (best\_idx == -1 || blocks[j].block\_size < blocks[best\_idx].block\_size) {  
 best\_idx = j;  
 }  
 }  
 }  
  
 if (best\_idx != -1) {  
 blocks[best\_idx].is\_free = 0;  
 printf("%d\t%d\t\t%d\t\t%d\n",   
 files[i].file\_no,   
 files[i].file\_size,   
 blocks[best\_idx].block\_no,   
 blocks[best\_idx].block\_size);  
 } else {  
 printf("%d\t%d\t\tNot Allocated\n", files[i].file\_no, files[i].file\_size);  
 }  
}

}

// Function for First Fit Allocation void firstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) { printf("\nMemory Management Scheme - First Fit\n"); printf("File\_no\tFile\_size\tBlock\_no\tBlock\_size\n");

for (int i = 0; i < n\_files; i++) {  
 int allocated = 0;  
  
 for (int j = 0; j < n\_blocks; j++) {  
 if (blocks[j].is\_free && blocks[j].block\_size >= files[i].file\_size) {  
 blocks[j].is\_free = 0;  
 printf("%d\t%d\t\t%d\t\t%d\n",   
 files[i].file\_no,   
 files[i].file\_size,   
 blocks[j].block\_no,   
 blocks[j].block\_size);  
 allocated = 1;  
 break;  
 }  
 }  
  
 if (!allocated) {  
 printf("%d\t%d\t\tNot Allocated\n", files[i].file\_no, files[i].file\_size);  
 }  
}

}

// Function for Worst Fit Allocation void worstFit(struct Block blocks[], int n\_blocks, struct File files[], int n\_files) { printf("\nMemory Management Scheme - Worst Fit\n"); printf("File\_no\tFile\_size\tBlock\_no\tBlock\_size\n");

for (int i = 0; i < n\_files; i++) {  
 int worst\_idx = -1;  
  
 for (int j = 0; j < n\_blocks; j++) {  
 if (blocks[j].is\_free && blocks[j].block\_size >= files[i].file\_size) {  
 if (worst\_idx == -1 || blocks[j].block\_size > blocks[worst\_idx].block\_size) {  
 worst\_idx = j;  
 }  
 }  
 }  
  
 if (worst\_idx != -1) {  
 blocks[worst\_idx].is\_free = 0;  
 printf("%d\t%d\t\t%d\t\t%d\n",   
 files[i].file\_no,   
 files[i].file\_size,   
 blocks[worst\_idx].block\_no,   
 blocks[worst\_idx].block\_size);  
 } else {  
 printf("%d\t%d\t\tNot Allocated\n", files[i].file\_no, files[i].file\_size);  
 }  
}

}

// Reset block availability before each allocation scheme void resetBlocks(struct Block original[], struct Block copy[], int n) { for (int i = 0; i < n; i++) { copy[i] = original[i]; } }

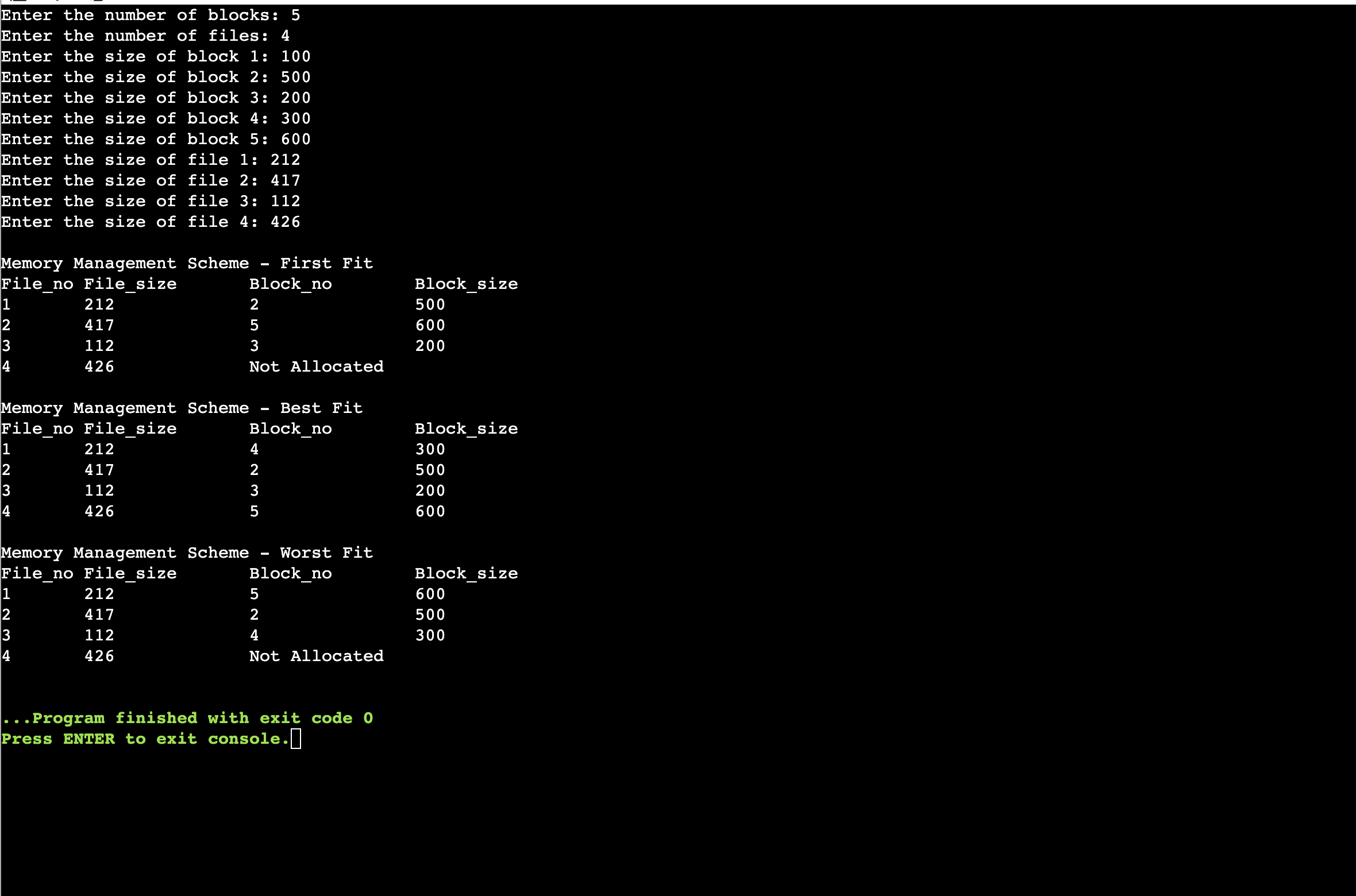
int main() { int n\_blocks, n\_files;

printf("Enter the number of blocks: ");  
scanf("%d", &n\_blocks);  
printf("Enter the number of files: ");  
scanf("%d", &n\_files);  
  
struct Block original\_blocks[n\_blocks];  
struct File files[n\_files];  
for (int i = 0; i < n\_blocks; i++) {  
 original\_blocks[i].block\_no = i + 1;  
 printf("Enter the size of block %d: ", i + 1);  
 scanf("%d", &original\_blocks[i].block\_size);  
 original\_blocks[i].is\_free = 1;  
}  
  
for (int i = 0; i < n\_files; i++) {  
 files[i].file\_no = i + 1;  
 printf("Enter the size of file %d: ", i + 1);  
 scanf("%d", &files[i].file\_size);  
}  
  
struct Block blocks[n\_blocks];  
  
resetBlocks(original\_blocks, blocks, n\_blocks);  
firstFit(blocks, n\_blocks, files, n\_files);  
  
resetBlocks(original\_blocks, blocks, n\_blocks);

bestFit(blocks, n\_blocks, files, n\_files);  
  
resetBlocks(original\_blocks, blocks, n\_blocks);  
worstFit(blocks, n\_blocks, files, n\_files);  
  
return 0;

}

OUTPUT:



Lab Program 7

Write a C program to stimulate page replacement algorithms

a)FIFO ,b)LRU ,c)Optimal

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

// Search if a page is in the frame int search(int key, int frame[], int frames) { for (int i = 0; i < frames; i++) if (frame[i] == key) return 1; return 0; }

// Find LRU page index int findLRU(int time[], int frames) { int min = time[0], pos = 0; for (int i = 1; i < frames; i++) { if (time[i] < min) { min = time[i]; pos = i; } } return pos; }

// Find page to replace using Optimal int findOptimal(int pages[], int frame[], int frames, int index, int n) { int farthest = index, pos = -1, i, j; for (i = 0; i < frames; i++) { for (j = index; j < n; j++) { if (frame[i] == pages[j]) { if (j > farthest) { farthest = j; pos = i; } break; } } if (j == n) return i; } return (pos == -1) ? 0 : pos; }

// FIFO Page Replacement void fifo(int pages[], int n, int frames) { int \*frame = (int \*)malloc(frames \* sizeof(int)); int page\_faults = 0, page\_hits = 0, index = 0; for (int i = 0; i < frames; i++) frame[i] = -1;

for (int i = 0; i < n; i++) {  
 if (!search(pages[i], frame, frames)) {  
 frame[index] = pages[i];  
 index = (index + 1) % frames;  
 page\_faults++;  
 } else {  
 page\_hits++;  
 }  
}  
  
printf("FIFO Page Faults: %d, Page Hits: %d\n", page\_faults, page\_hits);  
free(frame);

}

// LRU Page Replacement void lru(int pages[], int n, int frames) { int \*frame = (int \*)malloc(frames \* sizeof(int)); int \*recent = (int \*)malloc(frames \* sizeof(int)); int page\_faults = 0, page\_hits = 0; int time = 0;

for (int i = 0; i < frames; i++) {  
 frame[i] = -1;  
 recent[i] = -1;  
}  
  
for (int i = 0; i < n; i++) {  
 int page = pages[i];  
 int hit = 0;  
  
 // Check if page is already in frame  
 for (int j = 0; j < frames; j++) {  
 if (frame[j] == page) {  
 hit = 1;  
 page\_hits++;  
 recent[j] = time;  
 break;  
 }  
 }  
  
 // If not a hit, handle page fault  
 if (!hit) {  
 int pos = -1;  
  
 // Find empty frame  
 for (int j = 0; j < frames; j++) {  
 if (frame[j] == -1) {  
 pos = j;  
 break;  
 }  
 }  
  
 // If no empty frame, find least recently used  
 if (pos == -1)  
 pos = findLRU(recent, frames);  
  
 frame[pos] = page;  
 recent[pos] = time;  
 page\_faults++;  
 }  
  
 time++; // Increment time after each iteration  
}  
  
printf("LRU Page Faults: %d, Page Hits: %d\n", page\_faults, page\_hits);  
free(frame);  
free(recent);

}

// Optimal Page Replacement void optimal(int pages[], int n, int frames) { int \*frame = (int \*)malloc(frames \* sizeof(int)); int page\_faults = 0, page\_hits = 0;

for (int i = 0; i < frames; i++)  
 frame[i] = -1;  
  
for (int i = 0; i < n; i++) {  
 if (search(pages[i], frame, frames)) {  
 page\_hits++;  
 } else {  
 int pos = -1;  
 for (int j = 0; j < frames; j++) {  
 if (frame[j] == -1) {  
 pos = j;  
 break;  
 }  
 }  
 if (pos == -1)  
 pos = findOptimal(pages, frame, frames, i + 1, n);  
  
 frame[pos] = pages[i];  
 page\_faults++;  
 }  
}  
  
printf("Optimal Page Faults: %d, Page Hits: %d\n", page\_faults, page\_hits);  
free(frame);

}

// Main function int main() { int pages[100], n, frames;

printf("Enter the size of the pages:\n");  
scanf("%d", &n);  
  
printf("Enter the page strings:\n");  
for (int i = 0; i < n; i++)  
 scanf("%d", &pages[i]);  
  
printf("Enter the no of page frames:\n");  
scanf("%d", &frames);  
  
fifo(pages, n, frames);  
optimal(pages, n, frames);  
lru(pages, n, frames);  
return 0;  
 }

OUTPUT:

