6.1 Develop a C program to simulate the following contiguous memory allocation Techniques:

Worst fit

**Algorithm: Worst Fit Memory Allocation**

1. **Input Initialization**:
   * Read the number of memory blocks (numBlocks).
   * Read the number of files (numFiles).
   * Input the sizes of each block and file.
2. **Memory Allocation Process**:
   * For each file:
     1. Initialize a variable (largestBlockIndex) to track the block with the maximum leftover space after allocation.
     2. For each block:
        + Check if the block is unallocated and has enough space for the file.
        + If the block's space is greater than the current largestBlockIndex, update largestBlockIndex.
     3. Allocate the file to the block (largestBlockIndex) if a suitable block is found; otherwise, leave it unallocated.
     4. Update the block's fragmentation value and mark it as allocated.
3. **Output the Results**:
   * Display the file number, file size, allocated block number, block size, and fragmentation for each file.
   * If a file isn't allocated, display "Not Allocated."

#include <stdio.h>

#define MAX\_BLOCKS 25

void main() {

int fragmentation[MAX\_BLOCKS], blockSize[MAX\_BLOCKS], fileSize[MAX\_BLOCKS];

int i, j, numBlocks, numFiles, temp, largestBlockIndex;

int blockAllocated[MAX\_BLOCKS] = {0}, fileAllocation[MAX\_BLOCKS];

printf("\n\tMemory Management Scheme - Worst Fit");

printf("\nEnter the number of blocks: ");

scanf("%d", &numBlocks);

printf("Enter the number of files: ");

scanf("%d", &numFiles);

printf("\nEnter the size of the blocks:\n");

for (i = 1; i <= numBlocks; i++) {

printf("Block %d: ", i);

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

}

printf("Enter the size of the files:\n");

for (i = 1; i <= numFiles; i++) {

printf("File %d: ", i);

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

}

// Memory allocation using Worst Fit

for (i = 1; i <= numFiles; i++) {

largestBlockIndex = -1;

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

if (!blockAllocated[j] && blockSize[j] >= fileSize[i]) {

temp = blockSize[j] - fileSize[i];

if (largestBlockIndex == -1 || temp > (blockSize[largestBlockIndex] - fileSize[i])) {

largestBlockIndex = j;

}

}

}

if (largestBlockIndex != -1) {

fileAllocation[i] = largestBlockIndex;

fragmentation[i] = blockSize[largestBlockIndex] - fileSize[i];

blockAllocated[largestBlockIndex] = 1;

} else {

fileAllocation[i] = -1; // Not allocated

fragmentation[i] = -1;

}

}

// Output the results

printf("\nFile\_no:\tFile\_size:\tBlock\_no:\tBlock\_size:\tFragment");

for (i = 1; i <= numFiles; i++) {

if (fileAllocation[i] != -1) {

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",

i, fileSize[i], fileAllocation[i], blockSize[fileAllocation[i]], fragmentation[i]);

} else {

printf("\n%d\t\t%d\t\tNot Allocated", i, fileSize[i]);

}

}

}

**OUTPUT:**

Enter the number of blocks: 5

Enter the number of files: 4

Enter the size of the blocks:

Block 1: 200

Block 2: 300

Block 3: 400

Block 4: 500

Block 5: 600

Enter the size of the files:

File 1: 212

File 2: 417

File 3: 112

File 4: 426

File\_no: File\_size: Block\_no: Block\_size: Fragment

1 212 5 600 388

2 417 4 500 83

3 112 3 400 288

4 426 Not Allocated

6.2 Develop a C program to simulate the following contiguous memory allocation Techniques:

**Best fit**

Algorithm: Best Fit Memory Allocation

1. Input Initialization:
   * Read the number of memory blocks (numBlocks) and files (numFiles).
   * Input the sizes of the memory blocks and files.
2. Memory Allocation Process:
   * For each file:
     1. Initialize a variable (bestBlockIndex) to track the block with the minimum leftover space after allocation.
     2. Iterate over each block:
        + Check if the block is unallocated and has enough space for the file.
        + If the block’s leftover space (temp) is smaller than the current lowestFragmentation, update bestBlockIndex and lowestFragmentation.
     3. Allocate the file to the block (bestBlockIndex) if a suitable block is found; otherwise, leave it unallocated.
     4. Update the block's fragmentation and mark it as allocated.
3. Output the Results:
   * Display the file number, file size, allocated block number, block size, and fragmentation for each file.
   * If a file isn't allocated, display "Not Allocated."

#include <stdio.h>

#define MAX\_BLOCKS 25

void main() {

int fragmentation[MAX\_BLOCKS], blockSize[MAX\_BLOCKS], fileSize[MAX\_BLOCKS];

int i, j, numBlocks, numFiles, temp, lowestFragmentation;

int blockAllocated[MAX\_BLOCKS] = {0}, fileAllocation[MAX\_BLOCKS];

printf("\n\tMemory Management Scheme – Best Fit");

printf("\nEnter the number of blocks: ");

scanf("%d", &numBlocks);

printf("Enter the number of files: ");

scanf("%d", &numFiles);

printf("\nEnter the size of the blocks:\n");

for (i = 1; i <= numBlocks; i++) {

printf("Block %d: ", i);

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

}

printf("Enter the size of the files:\n");

for (i = 1; i <= numFiles; i++) {

printf("File %d: ", i);

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

}

// Memory allocation using Best Fit

for (i = 1; i <= numFiles; i++) {

int bestBlockIndex = -1;

lowestFragmentation = 10000; // Arbitrary high value

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

if (!blockAllocated[j] && blockSize[j] >= fileSize[i]) {

temp = blockSize[j] - fileSize[i];

if (temp < lowestFragmentation) {

bestBlockIndex = j;

lowestFragmentation = temp;

}

}

}

if (bestBlockIndex != -1) {

fileAllocation[i] = bestBlockIndex;

fragmentation[i] = blockSize[bestBlockIndex] - fileSize[i];

blockAllocated[bestBlockIndex] = 1;

} else {

fileAllocation[i] = -1; // Not allocated

fragmentation[i] = -1;

}

}

// Output the results

printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragment");

for (i = 1; i <= numFiles; i++) {

if (fileAllocation[i] != -1) {

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",

i, fileSize[i], fileAllocation[i], blockSize[fileAllocation[i]], fragmentation[i]);

} else {

printf("\n%d\t\t%d\t\tNot Allocated", i, fileSize[i]);

}

}

}

**OUTPUT:**

Enter the number of blocks: 5

Enter the number of files: 4

Enter the size of the blocks:

Block 1: 100

Block 2: 500

Block 3: 200

Block 4: 300

Block 5: 600

Enter the size of the files:

File 1: 212

File 2: 417

File 3: 112

File 4: 426

File No File Size Block No Block Size Fragment

1 212 4 300 88

2 417 5 600 183

3 112 3 200 88

4 426 Not Allocated

6.3 Develop a C program to simulate the following contiguous memory allocation Techniques:

**First fit**

Algorithm: First Fit Memory Allocation

1. Input Initialization:
   * Read the number of memory blocks (numBlocks) and files (numFiles).
   * Input the sizes of each memory block and file.
2. Memory Allocation Process:
   * For each file:
     1. Iterate over the blocks to find the first unallocated block that fits the file size.
     2. If a suitable block is found:
        + Allocate the block to the file.
        + Calculate the fragmentation.
        + Mark the block as allocated.
     3. If no suitable block is found, the file remains unallocated.
3. Output the Results:
   * Display the file number, file size, allocated block number, block size, and fragmentation for each file.

**PROGRAM:**

#include <stdio.h>

#define MAX\_BLOCKS 25

void main() {

int fragmentation[MAX\_BLOCKS], blockSize[MAX\_BLOCKS], fileSize[MAX\_BLOCKS];

int i, j, numBlocks, numFiles, temp;

int blockAllocated[MAX\_BLOCKS] = {0}, fileAllocation[MAX\_BLOCKS] = {0};

printf("\n\tMemory Management Scheme - First Fit");

printf("\nEnter the number of blocks: ");

scanf("%d", &numBlocks);

printf("Enter the number of files: ");

scanf("%d", &numFiles);

printf("\nEnter the size of the blocks:\n");

for (i = 1; i <= numBlocks; i++) {

printf("Block %d: ", i);

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

}

printf("Enter the size of the files:\n");

for (i = 1; i <= numFiles; i++) {

printf("File %d: ", i);

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

}

// Memory allocation using First Fit

for (i = 1; i <= numFiles; i++) {

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

if (!blockAllocated[j]) { // Check if the block is not allocated

temp = blockSize[j] - fileSize[i];

if (temp >= 0) {

fileAllocation[i] = j;

fragmentation[i] = temp;

blockAllocated[j] = 1; // Mark block as allocated

break;

}

}

}

if (!fileAllocation[i]) { // If no block found

fragmentation[i] = -1; // Indicate no allocation

}

}

// Output the results

printf("\nFile\_no:\tFile\_size:\tBlock\_no:\tBlock\_size:\tFragment");

for (i = 1; i <= numFiles; i++) {

if (fileAllocation[i] != 0) {

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",

i, fileSize[i], fileAllocation[i], blockSize[fileAllocation[i]], fragmentation[i]);

} else {

printf("\n%d\t\t%d\t\tNot Allocated", i, fileSize[i]);

}

}

}

**OUTPUT:**

Enter the number of blocks: 5

Enter the number of files: 4

Enter the size of the blocks:

Block 1: 100

Block 2: 500

Block 3: 200

Block 4: 300

Block 5: 600

Enter the size of the files:

File 1: 212

File 2: 417

File 3: 112

File 4: 426

File\_no: File\_size: Block\_no: Block\_size: Fragment

1 212 2 500 288

2 417 5 600 183

3 112 3 200 88

4 426 Not Allocated