PAGE NO:
DATE:

Q) Write a C program to simulate the following contiguous memory allocation techniques
a) Worst-fit
b) Best-fit
c) First-fit

```
#include <stdio.h>
#include <stdlib.h>
# define MAX_PARTITIONS 10
# define MAX_PROCESSES 10

typedef struct {
    int size;
    int isAllocated;
} partition;

typedef struct {
    int size;
    int isAllocated;
    int partitionIndex;
} process;

void allocate(Process processes[], int pro
-&unt, Partition partitions[], int partition
-nt, int (*findIndex)(Partition [], int, in
int totalFragmentation=0;
printf("\n File NO, \t process size \t
    Block NO. \t Block size \t Fragme
for (int i=0; i<ProcessCount; i++) {
    int index= findIndex(Partitions, partitio
        processes[i].size);
    if (index != -1) {
```

```
        process[i].isAllocated = 1;
        process[i].partitionIndex = index;
        partitions[index].isAllocated = 1;
        int fragmentation = partitions[index].size -
            process[i].size;
        totalFragmentation += fragmentation;
        printf("%d \t %d \t %d \t %d \t %d \n",
            i, process[i].size, index, partitions[index].
            size, fragmentation);
    } else {
        printf("%d \t %d \t \t Process could
            not be allocated \n", i, ~~process[i]~~
            processes[i].size);
    }
}

printf("\n Total Fragmentation: %d \n",
    totalFragmentation);
}

int firstFitIndex(partition partitions[], int
    partitionCount, int processSize) {
    for(int i=0; i < partitionCount; i++) {
        if(!partitions[i].isAllocated &&
            partitions[i].size >= processSize) {
            return i;
        }
    }
    return -1;
}

int ~~best~~ worst FitIndex(partition partitions[], int
    partitionCount, int processSize) {
    int worstIndex = -1;
    for(int i=0; i<partitionCount; i++) {
        if(!partitions[i].isAllocated &&
```

```
        partitions[i].size >= processSize) {
        if (worstIndex == -1 || partition[i].size
            > partition[worstIndex].size) {
            worstIndex = i;
        }
    }
}
return worstIndex;
}

int bestFitIndex(partition partitions[], int
    partitionCount, int processSize) {
    int bestIndex = -1;
    for(int i=0; i<partitionCount;
        if(!partitions[i].isAllocated &&
            partitions[i].size >= processSize){
        if(bestIndex == -1 || partitions[i].size
            < partitions[bestIndex].size){
            bestIndex = i;
        }
    }
}
return bestIndex;
}
void reset(partition partitions[], int partition
    Count, process processes[], int processCount){
    for(int i=0; i<partitionCount; i++)
        partitions[i].isAllocated = 0;
    for(int i=0; i<processCount; i++){
        process[i].isAllocated = 0;
        process[i].partitionIndex = -1;
    }
}
```

```
int main() {
    int partitionCount, processCount, choice,
    alreadyAllocated;
    Partition partitions[MAX_PARTITIONS];
    Process processes[MAX_PROCESSES];
    printf("Enter the number of partitions:");
    scanf("%d", &partitionCount);
    printf("Enter size of each partition:\n");
    for(int i=0; i<partitionCount; i++) {
        printf("partition %d: ", i);
        scanf("%d", &partitions[i].size);
        partitions[i].isAllocated = 0;
    }
    printf("Enter no. of Processes: ");
    scanf("%d", &processCount);
    printf("Enter the size of each process:\n");
    for(int i=0; i<processCount; i++) {
        printf("Process %d ", i);
        scanf("%d", &processes[i].size);
        processes[i].isAllocated = 0;
        processes[i].partitionIndex = -1;
    }
    printf("Choose memory allocation
        strategy: \n1. FirstFit \n2.
    Beast-Fit \n3. Worst_Fit \n");
    scanf("%d", &choice);
    int (*allocationStrategy)(Partition[], int,
        int) = NULL;
    switch(choice) {
    case 1: allocationStrategy = firstFitIndex;
        break;
    case 2: allocationStrategy = bestFitIndex;
        break;
```

```
case 3: allocationStrategy = WORST_FIT;
        break;
default: printf("Invalid Choice");
        return;
}
printf("Memory Management Scheme");
printf("File no\t File size\t Block no\t Block size\t Fragment");
allocate(processes, process count, partitions, partition count, allocationStrategy);
}
```

output

```
Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of blocks:
Block1: 5
Block2: 2
Block3: 7
Enter the size of files:
File 1: 1
File 2: 4
```

Memory Management scheme - First Fit

File no:	File Size	Block no:	Block size	Fragment
1	1	1	5	4
2	4	3	7	3

memory management scheme - worst fit

File no	File_size	Block_no	Block size	Fragment
1	1	3	7	6
2	4	1	5	1

PAGE NO:
DATE:

memory management Scheme - Best Fit

File no	File size	Block no	Block size	Fragmentation
1	1	2	3	2
2	4	1	5	1