Write a C program to simulate the following contiguous memory allocation technique

a) Worst - fit
b) Best fit
c) First fit

a. Worst fit

```
#include <stdio.h>
void worstfit (int blocksize[], int blocks,
int process[],
  int process;
  { int allocation [process];
      int occupied [blocks];
  for (int i=0; i<process; i++)
      { allocation [i] = -1;
  }
  for (int i=0; i<process; i++)
  { int indexPlaced = -1;
    for (int j=0; j<blocks; j++)
  { if (blockSize [j] >= processSize [i]
        && !occupied [j])
    {
      if (indexPlaced == -1)
        indexPlaced = j;
  else if (blockSize [indexPlaced] < blockSize[j])
      indexPlaced = j;
  }
}
```

```
if (indexPlaced != -1)
{ allocation[i] = indexPlaced;
  occupied[indexPlaced] = 1;
  blockSize[indexPlaced] = processSize[i];
  }
}

printf("\n Process No.\t Process Size\t
      block no:\n");
for(int i=0; i<Process; i++)
{ printf("%d\t\t\t %d\t\t\t", i+1,
    process Size[i]);
  if (allocation[i] != -1)
    printf("%d\n", allocation[i]+1);
else
  printf("Not allocated\n");
}

main()
{ int i, blocks, process;
  printf("Enter the no. of blocks:");
  scanf("%d", &blocks);
int blockSize[blocks];
  printf("\n Enter size of each block:");
  for(int i=0; i<blocks; i++);
  scanf("%d", &process);
  int processSize[process];{
  printf("\n Enter size of each process:");
  for(i=0; i<process; i++)
  scanf("%d", &processSize[i]);
  worstfit(blockSize, blocks, processSize, proc);
  return 0;
}
```

Output

Enter no. of blocks : 3
Enter size of each block : 5 2 7
Enter no. of process : 2
Enter size of each process : 1 4

Process No:	Process size	Block no:
1	1	3
2	4	1

b:
```
# include <stdio.h>
# define MAX
void BestFit (int blockSize [], int blocks,
int procSize [], int process, int m)
{ int allocation [process];
  int occupied [blocks];
for (int i = 0; i < process; i++)
  { allocation [i] = -1;
  }
for (int i = 0; i < process; i++)
{ int indexPlaced = -1;
for (int j = 0; j < blocks; j++)
{ if (blockSize [j] >= procSize [i]
  && !occupied [j])
  { if (indexPlaced == -1)
    indexPlaced = j;
  else if (blockSize [j] < blockSize [indexPlaced])
    indexPlaced = j;
  }
}
```

```
if (indexPlaced[i] != -1)
{
  allocation[i] = indexPlaced[i];
  occupied[indexPlaced[i]] = 1;
}
}
printf("\n Process No\t Process Size\t Block no\n");
for(int i=0; i<process; i++)
{ printf("%d\t\t%d\t\t", i+1, processSize[i]);
  if (allocation[i] != -1)
printf("%d\n", allocation[i]+1);
else
  printf("Not allocated\n");
}

int main()
{
  int p, m, j;
  printf("Enter the number of process and
                blocks:");
scanf("%d%d", &p, &m);
int processSize[p], blockSize[m];
printf("Enter the process size:");
  for(j=0; j<p; j++)
scanf("%d", &processSize[j]);
printf("Enter the Block sizes:");
  for(j=0; j<m; j++)
  scanf("%d", &blockSize[j]);
int blocks = sizeof(BlockSize)/sizeof(blockSize[0]);
int process = sizeof(processSize)/sizeof(processSize[0]);
Best Fit(blockSize, blocks, processSize, process, m);
```

Date __/__/__
Page _____

return 0;
}

Enter the number of process and blocks:
2 3
Enter the process size: 1 4
Enter the block size: 5 2 7

Process No	Process Size	Block no:
1	1	2
2	4	1

c- first fit

```
#include <stdio.h>
#include <conio.h>
#define max 25

void main()
{ int frag[max], b[max], f[max], i, j,
nb, nf; static int bf[max], ff[max];
printf("\n\tmemory management scheme -
                first fit");
printf("\n Enter the number of blocks:");
scanf("%d%d", &nb);
printf("Enter the number of files:");
scanf("%d", &nf);
printf("\n Enter the size of blocks :-\n");
for(i=1; i<=nb; i++)
  { printf("Block %d:", i);
scanf("%d", &b[i]);
}
```

```
printf("\n Enter the size of the files :-\n");
    for (i=1; i<nf; i++)
     printf("File %d", i);
     scanf("%d", &f[i]);
    }

printf("\n File-no:\t File-size:\t
          Block-no:\t Block-size:");
for(i=1; i<=nf; i++)
  { int allocated=0;
    for(j=1; j<=nb; j++)
      if(bf[j]!=1)
      { if(ff[i]<=f[i])
          bf[j]=1;
          frag[i]=b[j]-f[i];
          allocated=1;
printf("\n%d\t%d\t\t%d\t\t%d\t\t%d",
          i, f[i], ff[i], bff[i]));
          break;
      }
  }
if(!allocated)
{ printf("\n%d\t\t\t%d\t\t\tNot
    Allocated\t\t-", i, f[i]));
}
getch();
```

Output

Memory Management Scheme: First

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:

Blocks 1: 5

Blocks 2: 2

Blocks 3: 7

Enter the size of the files:

Files 1: 1

Files 2: 4

File no	File size	Block no	Block size
1	1	1	5
2	4	3	7

```
       Memory Management Scheme - Worst Fit
Enter the number of blocks:3
Enter the number of files:5 2 7

Enter the size of the blocks:
Block 1:Block 2:Block 3:1 4 2
Enter the size of the files:
File 1:File 2:File 3:2 4 5
File 4:File 5:
File_no:        File_size:      Block_no:       Block_size:     Fragment
1               4               2               7               3
2               2               1               2               0
3               2               Not Allocated   -               -1
4               4               Not Allocated   -               -3
5               5               Not Allocated   -               -4
Process returned 5 (0x5)   execution time : 44.315 s
Press any key to continue.
```

```
Enter the number of blocks:2
Enter the number of files:2

Enter the size of the blocks:
Block 1:5
Block 2:3
Enter the size of the files:
File 1:2
File 2:1

File No File Size       Block No        Block Size      Fragment
1               2               2               3               1
2               1               1               5               4
Process returned 2 (0x2)   execution time : 10.630 s
Press any key to continue.
```

```
       Memory Management Scheme - Worst Fit
Enter the number of blocks:2
Enter the number of files:2

Enter the size of the blocks:
Block 1:6
Block 2:5
Enter the size of the files:
File 1:3
File 2:2

File_no:        File_size:      Block_no:       Block_size:     Fragement
1               3               1               6               3
2               2               2               5               3
Process returned 2 (0x2)   execution time : 9.537 s
Press any key to continue.
```