Program 6. Develop a C program to simulate the following contiguous memory allocation Techniques: a)Worst fit b)Best fit c)First fit.

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

void allocate(int b[], int f[], int nf, int nb, char* method);

int main() {
    int b[max], f[max], nb, nf, i;
    char method[10];

    printf("Enter the number of blocks: ");
    scanf("%d", &nb);
    printf("Enter the number of files: ");
    scanf("%d", &nf);

    printf("Enter the size of the blocks:\n");
    for(i = 0; i < nb; i++) {
        printf("Block %d: ", i+1);
        scanf("%d", &b[i]);
    }

    printf("Enter the size of the files:\n");
    for(i = 0; i < nf; i++) {
        printf("File %d: ", i+1);
        scanf("%d", &f[i]);
    }

    printf("Enter allocation method (worst, best, first): ");
    scanf("%s", method);

    allocate(b, f, nf, nb, method);

    return 0;
}

void allocate(int b[], int f[], int nf, int nb, char* method) {
    int bf[max], ff[max];
    int i, j, temp, highest, lowest, index;

    for(i = 0; i < nf; i++) {
        ff[i] = -1;
    }

    for(i = 0; i < nb; i++) {
```

```
    bf[i] = 0;
}

for(i = 0; i < nf; i++) {
    index = -1;
    if (strcmp(method, "worst") == 0) {
        highest = -1;
        for(j = 0; j < nb; j++) {
            if(bf[j] == 0 && b[j] >= f[i]) {
                temp = b[j] - f[i];
                if(temp > highest) {
                    highest = temp;
                    index = j;
                }
            }
        }
    } else if (strcmp(method, "best") == 0) {
        lowest = max;
        for(j = 0; j < nb; j++) {
            if(bf[j] == 0 && b[j] >= f[i]) {
                temp = b[j] - f[i];
                if(temp < lowest) {
                    lowest = temp;
                    index = j;
                }
            }
        }
    } else if (strcmp(method, "first") == 0) {
        for(j = 0; j < nb; j++) {
            if(bf[j] == 0 && b[j] >= f[i]) {
                index = j;
                break;
            }
        }
    }

    if(index != -1) {
        ff[i] = index;
        bf[index] = 1;
    }
}

printf("\nFile_no\tFile_size\tBlock_no\tBlock_size\tFragment\n");
for(i = 0; i < nf; i++) {
    printf("%d\t\t%d\t\t", i+1, f[i]);
    if(ff[i] != -1) {
```

```
            printf("%d\t\t%d\t\t%d\n", ff[i]+1, b[ff[i]], b[ff[i]] - f[i]);
        } else {
            printf("Not Allocated\n");
        }
    }
}
```

Output:

```
krishna@ubuntu:~$ ./a.out
Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of the blocks:
Block 1: 12
Block 2: 15
Block 3: 10
Enter the size of the files:
File 1: 10
File 2: 8
Enter allocation method (worst, best, first): best

File_no File_size       Block_no        Block_size      Fragment
1               10              3               10              0
2               8               1               12              4
```

```
krishna@ubuntu:~$ ./a.out
Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of the blocks:
Block 1: 12
Block 2: 15
Block 3: 10
Enter the size of the files:
File 1: 10
File 2: 8
Enter allocation method (worst, best, first): worst

File_no File_size       Block_no        Block_size      Fragment
1               10              2               15              5
2               8               1               12              4
```

```
krishna@ubuntu:~$ ./a.out
Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of the blocks:
Block 1: 12
Block 2: 15
Block 3: 10
Enter the size of the files:
File 1: 10
File 2: 8
Enter allocation method (worst, best, first): first

File_no File_size       Block_no        Block_size      Fragment
1               10              1               12              2
2               8               2               15              7
```