LAB 09

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

ANSWER:

CODE:

```
#include <stdio.h>

#define MAX 10

int p[MAX], np, b[MAX], nb, c[MAX], d[MAX], alloc[MAX], flag[MAX];

// Function prototypes
void input();
void first_fit();
void best_fit();
void worst_fit();

int main() {
   int ch;

   printf("\n**** Memory Allocation Strategies ****\n");
   input();

   if (np <= nb) {
      do {
         printf("\n1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit");
         printf("\nEnter your choice: ");
         scanf("%d", &ch);

         switch (ch) {
            case 1: first_fit(); break;
            case 2: best_fit(); break;
            case 3: worst_fit(); break;
            case 4: printf("\nExiting...\n"); break;
            default: printf("\nInvalid Choice...!\n"); break;
         }
      } while (ch < 4);
   } else {
      printf("\nNumber of processes exceeds number of memory blocks. Allocation not
possible.\n");
   }

   return 0;
```

```
}

void input() {
    int i, j;

    printf("\nEnter the number of processes: ");
    scanf("%d", &np);

    printf("\nEnter the number of blocks: ");
    scanf("%d", &nb);

    printf("\nEnter the size of each process:\n");
    for (i = 0; i < np; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d", &p[i]);
    }

    printf("\nEnter the block sizes:\n");
    for (j = 0; j < nb; j++) {
        printf("Block %d: ", j + 1);
        scanf("%d", &b[j]);
        c[j] = b[j];  // Copy for Best Fit
        d[j] = b[j];  // Copy for Worst Fit
    }
}

// First Fit Allocation Strategy
void first_fit() {
    printf("\nFirst Fit Allocation\n");

    for (int i = 0; i < np; i++) {
        flag[i] = 1;
        for (int j = 0; j < nb; j++) {
            if (p[i] <= b[j]) {
                alloc[j] = p[i];
                printf("\nProcess %d of size %d is allocated in Block %d of size %d\n", i + 1, p[i], j
+ 1, b[j]);
                b[j] = 0; // Mark block as allocated
                flag[i] = 0;
                break;
            }
        }
        if (flag[i] != 0) {
            printf("\nProcess %d of size %d is not allocated\n", i + 1, p[i]);
        }
    }
}
```

```
// Best Fit Allocation Strategy
void best_fit() {
    int i, j, temp;

    printf("\nBest Fit Allocation\n");

    // Sorting block sizes in ascending order
    for (i = 0; i < nb - 1; i++) {
        for (j = i + 1; j < nb; j++) {
            if (c[i] > c[j]) {
                temp = c[i];
                c[i] = c[j];
                c[j] = temp;
            }
        }
    }

    printf("\nAfter sorting block sizes:\n");
    for (i = 0; i < nb; i++) {
        printf("Block %d: %d\n", i + 1, c[i]);
    }

    for (i = 0; i < np; i++) {
        flag[i] = 1;
        for (j = 0; j < nb; j++) {
            if (p[i] <= c[j]) {
                alloc[j] = p[i];
                printf("\nProcess %d of size %d is allocated in Block %d of size %d\n", i + 1, p[i], j
+ 1, c[j]);
                c[j] = 0; // Mark block as allocated
                flag[i] = 0;
                break;
            }
        }
        if (flag[i] != 0) {
            printf("\nProcess %d of size %d is not allocated\n", i + 1, p[i]);
        }
    }
}

// Worst Fit Allocation Strategy
void worst_fit() {
    int i, j, temp;

    printf("\nWorst Fit Allocation\n");

    // Sorting block sizes in descending order
    for (i = 0; i < nb - 1; i++) {
```

```
        for (j = i + 1; j < nb; j++) {
            if (d[i] < d[j]) {
                temp = d[i];
                d[i] = d[j];
                d[j] = temp;
            }
        }
    }

    printf("\nAfter sorting block sizes:\n");
    for (i = 0; i < nb; i++) {
        printf("Block %d: %d\n", i + 1, d[i]);
    }

    for (i = 0; i < np; i++) {
        flag[i] = 1;
        for (j = 0; j < nb; j++) {
            if (p[i] <= d[j]) {
                alloc[j] = p[i];
                printf("\nProcess %d of size %d is allocated in Block %d of size %d\n", i + 1, p[i], j
+ 1, d[j]);
                d[j] = 0; // Mark block as allocated
                flag[i] = 0;
                break;
            }
        }
        if (flag[i] != 0) {
            printf("\nProcess %d of size %d is not allocated\n", i + 1, p[i]);
        }
    }
}
```

OUTPUT:

```
**** Memory Allocation Strategies ****

Enter the number of processes: 4

Enter the number of blocks: 5

Enter the size of each process:
Process 1: 100
Process 2: 400
Process 3: 200
Process 4: 500

Enter the block sizes:
Block 1: 600
Block 2: 500
Block 3: 300
Block 4: 200
Block 5: 100

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 1

First Fit Allocation

Process 1 of size 100 is allocated in Block 1 of size 600

Process 2 of size 400 is allocated in Block 2 of size 500

Process 3 of size 200 is allocated in Block 3 of size 300

Process 4 of size 500 is not allocated
```

```
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 2

Best Fit Allocation

After sorting block sizes:
Block 1: 100
Block 2: 200
Block 3: 300
Block 4: 500
Block 5: 600

Process 1 of size 100 is allocated in Block 1 of size 100

Process 2 of size 400 is allocated in Block 4 of size 500

Process 3 of size 200 is allocated in Block 2 of size 200

Process 4 of size 500 is allocated in Block 5 of size 600

1. First Fit
```

```
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 3

Worst Fit Allocation

After sorting block sizes:
Block 1: 600
Block 2: 500
Block 3: 300
Block 4: 200
Block 5: 100

Process 1 of size 100 is allocated in Block 1 of size 600

Process 2 of size 400 is allocated in Block 2 of size 500

Process 3 of size 200 is allocated in Block 3 of size 300

Process 4 of size 500 is not allocated

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 4

Exiting...
```