Name: Subhadeep Chell
Reg.No: 21BCE1288

OPERATING SYSTEMS:

>> LAB EXCERCISE-11: Contiguous memory allocation Exercise.

Q.Ans.

The required memory partition calculated and allocated according to First-fit, Best-fit and Worst-fit algorithms is given as :

Given -> memory partitions of 100K, 500K, 200K, 300K, and 600K (in order)

Therefore,

First-fit algorithm:

- **212K process: allocated to the first partition (100K)**
- **417K process: allocated to the fourth partition (300K)**
- **112K process: allocated to the first partition (88K remaining)**
- **426K process: allocated to the fifth partition (174K remaining)**

Best-fit algorithm:

- **212K process: allocated to the second partition (288K remaining)**
- **417K process: allocated to the fifth partition (183K remaining)**
- **112K process: allocated to the third partition (88K remaining)**
- **426K process: allocated to the fifth partition (0K remaining)**

Worst-fit algorithm:

- **212K process: allocated to the fifth partition (388K remaining)**
- **417K process: allocated to the fifth partition (0K remaining)**
- **112K process: allocated to the second partition (88K remaining)**
- **426K process: allocated to the fifth partition (0K remaining)**

So, we can already see, that the Best-fit algorithm makes the most efficient use of memory because it allocates processes to the partition with the smallest amount of free space that is still large enough to hold the process.

The required C program for Best-fit algorithm is given as:

```
#include <stdio.h>

#define NUM_PARTITIONS 5
#define NUM_PROCESSES 4

int partitions[NUM_PARTITIONS] = {100, 500, 200, 300, 600};
int processes[NUM_PROCESSES] = {212, 417, 112, 426};

void best_fit() {
    int allocation[NUM_PROCESSES] = {-1};

    for (int i = 0; i < NUM_PROCESSES; i++) {
        int best_index = -1;
        for (int j = 0; j < NUM_PARTITIONS; j++) {
            if (partitions[j] >= processes[i]) {
                if (best_index == -1 || partitions[j] < partitions[best_index])
{
                    best_index = j;
                }
            }
        }

        if (best_index != -1) {
            allocation[i] = best_index;
            partitions[best_index] -= processes[i];
        }
    }
```

```
    printf("Process\tSize\tPartition\n");
    for (int i = 0; i < NUM_PROCESSES; i++) {
        printf("%d\t%d\t", i, processes[i]);
        if (allocation[i] != -1) {
            printf("%d\n", allocation[i]);
        } else {
            printf("Not allocated\n");
        }
    }
}

int main() {
    best_fit();
    return 0;
}
```

In the above program, we implement the Best-fit algorithm by iterating through each process and finding the partition with the smallest amount of free space that is still large enough to hold the process. It then updates the allocation array and the remaining space in the partition array. Finally, it prints out a table of the process, its size, and the partition it was allocated to (or "Not allocated" if it couldn't be allocated).
