A.Y.: 2022-23 **Class: S.Y.B.Tech** **Sub:** System Fundamentals

Experiment 6

(Contiguous Memory Allocation Policy)

Aim: Implement Best Fit, First Fit and Worst Fit Memory allocation policy.

Theory: Memory manager of operating system efficiently manages the primary memory of the computer. Since every process must have some amount of primary memory in order to execute, the performance of the memory manager is crucial to the performance of the entire system. Managing the sharing of primary memory and minimizing memory access time are the basic goals of the memory manager."

The real challenge of efficiently managing memory is seen in the case of a system which has multiple processes running at the same time. Since primary memory can be space-multiplexed, the memory manager can allocate a portion of primary memory to each process for its own use. However, the memory manager must keep track of which processes are running in which memory locations, and it must also determine how to allocate and deallocate available memory when new processes are created and when old processes complete execution. In contiguous memory management strategy memory manager allocates space to processes competing for memory, three of the most popular are Best fit, Worst fit, and First fit. Each of these strategies are described below

First Fit

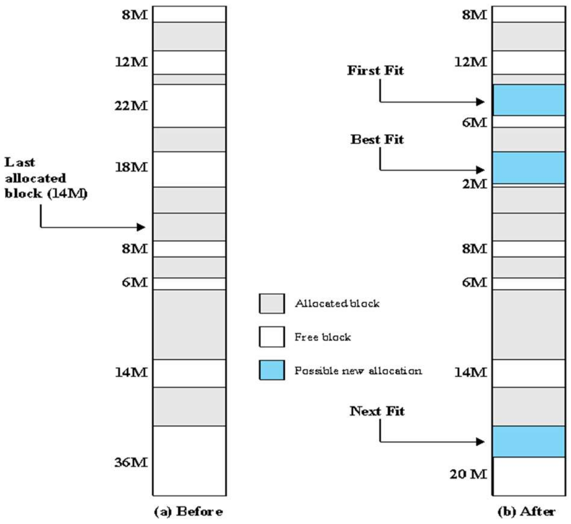
In the first fit approach is to allocate the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition.

Best Fit

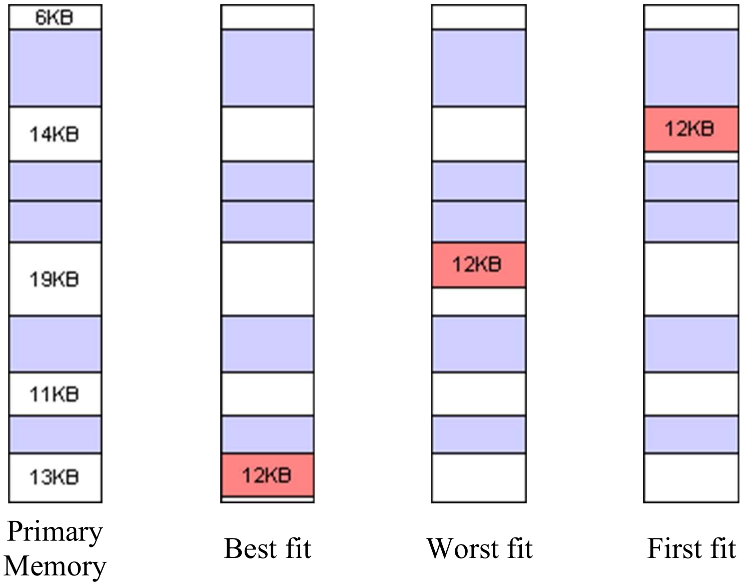
The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed.

Worst fit

In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.



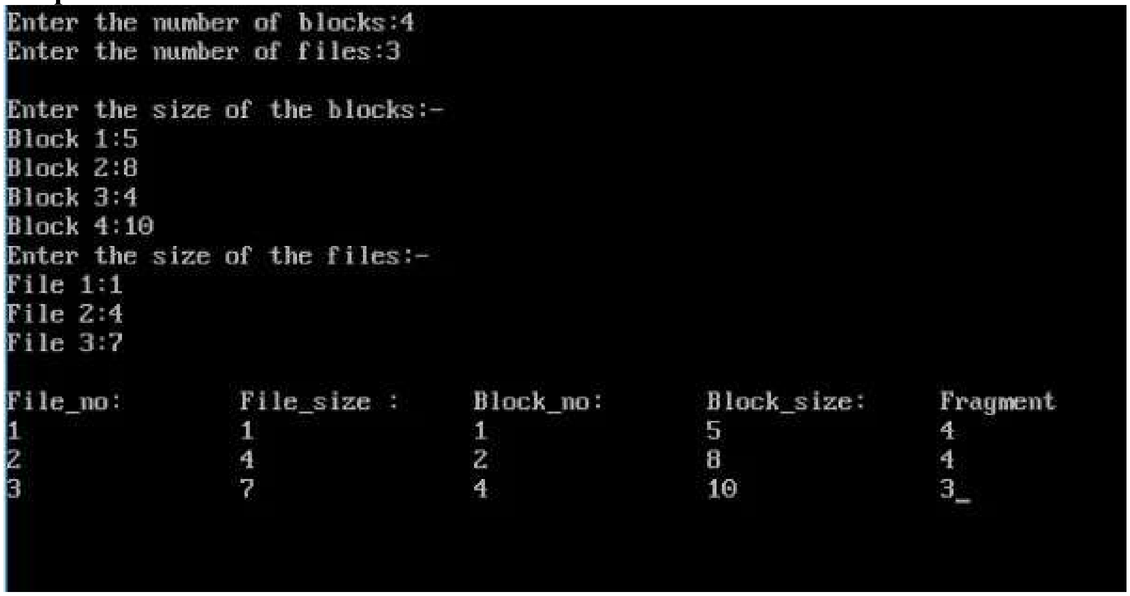
Example Memory Configuration before and after of 16-Mbyte Block



The Best fit and First fit strategies both leave a tiny segment of memory unallocated just beyond the new process. Since the amount of memory is small, it is not likely that any new processes can be loaded here. This condition of splitting primary memory into segments as the memory is allocated and deallocated is known as *fragmentation*. The Worst fit strategy attempts to reduce the problem of fragmentation by allocating the largest fragments to new processes. Thus, a larger amount of space will be left as seen in the diagram above.

Program Outcome for First Fit

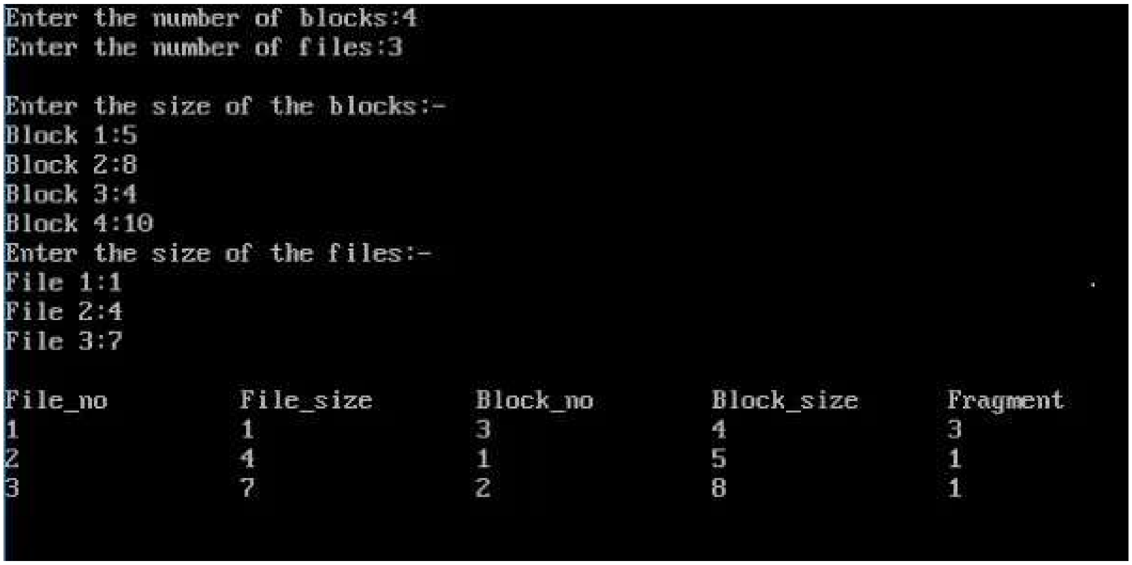
Output



```
Enter the number of blocks:4
Enter the number of files:3

Enter the size of the blocks:-
Block 1:5
Block 2:8
Block 3:4
Block 4:10
Enter the size of the files:-
File 1:1
File 2:4
File 3:7

File_no:        File_size :     Block_no:       Block_size:     Fragment
1               1               1               5               4
2               4               2               8               4
3               7               4               10              3_
```

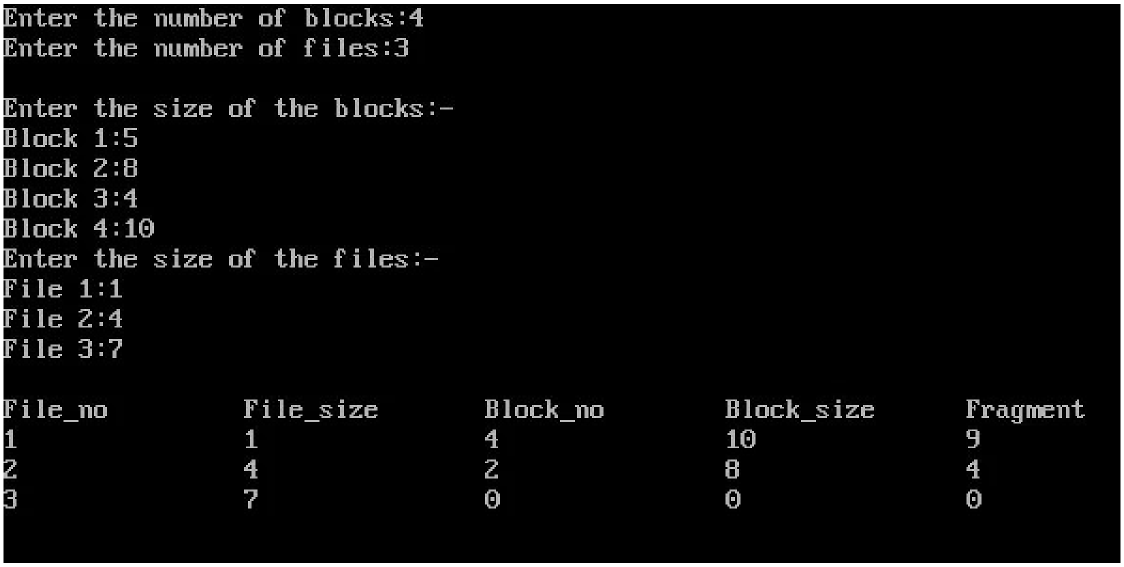
Program Outcome for Best Fit

Output



```
Enter the number of blocks:4
Enter the number of files:3

Enter the size of the blocks:-
Block 1:5
Block 2:8
Block 3:4
Block 4:10
Enter the size of the files:-
File 1:1
File 2:4
File 3:7

File_no         File_size       Block_no        Block_size      Fragment
1               1               3               4               3
2               4               1               5               1
3               7               2               8               1
```

Program outcome for Worst Fit



```
Enter the number of blocks:4
Enter the number of files:3

Enter the size of the blocks:-
Block 1:5
Block 2:8
Block 3:4
Block 4:10
Enter the size of the files:-
File 1:1
File 2:4
File 3:7

File_no         File_size       Block_no        Block_size      Fragment
1               1               4               10              9
2               4               2               8               4
3               7               0               0               0
```

Lab Assignment to be done by students:

Note: Reuse Left out space

Input:

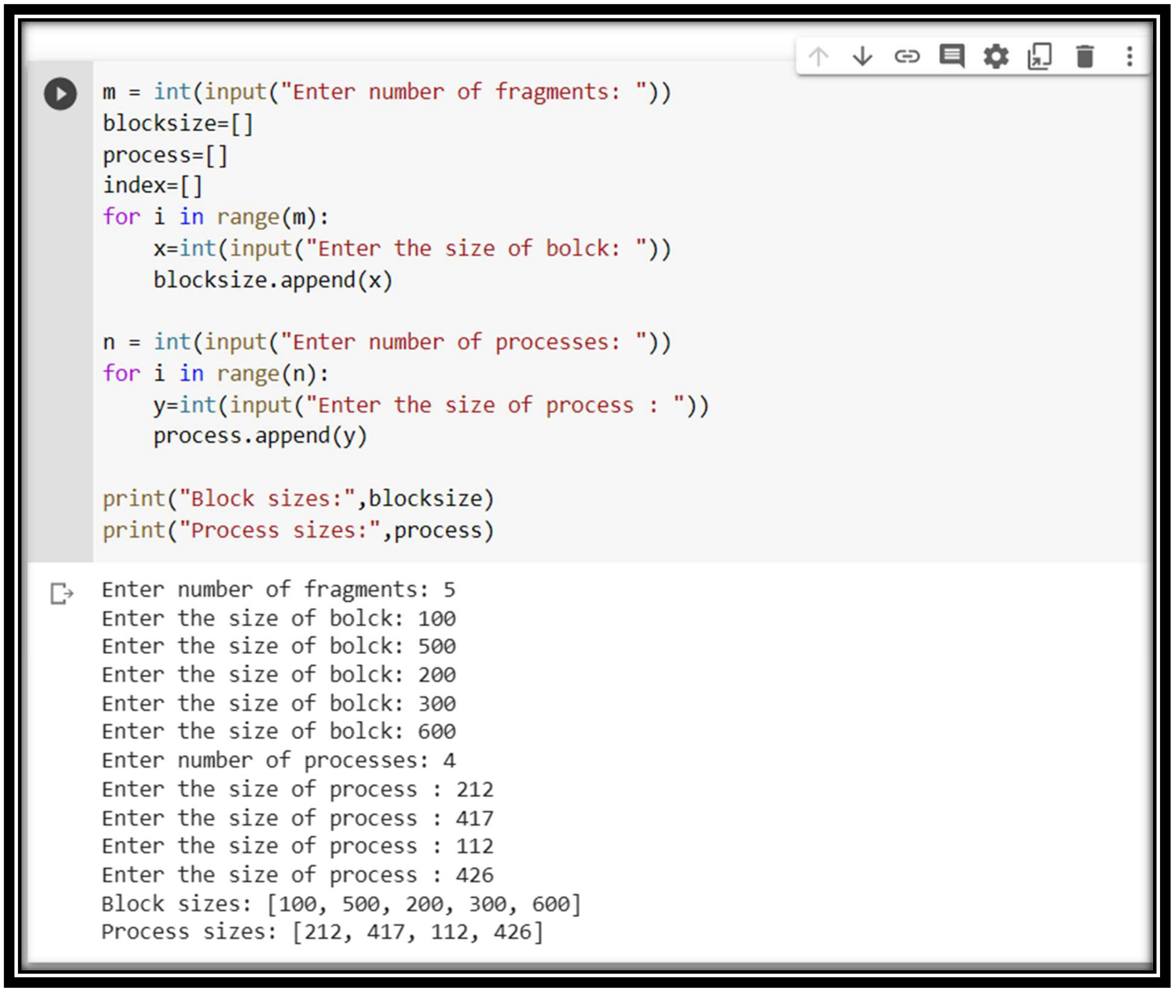
blockSize[] = {100, 500, 200, 300, 600};

Process Size[] = {212, 417, 112, 426};

Output:

Process No.	Process Size	Block no.
1	212	4
2	417	2
3	112	3
4	426	5

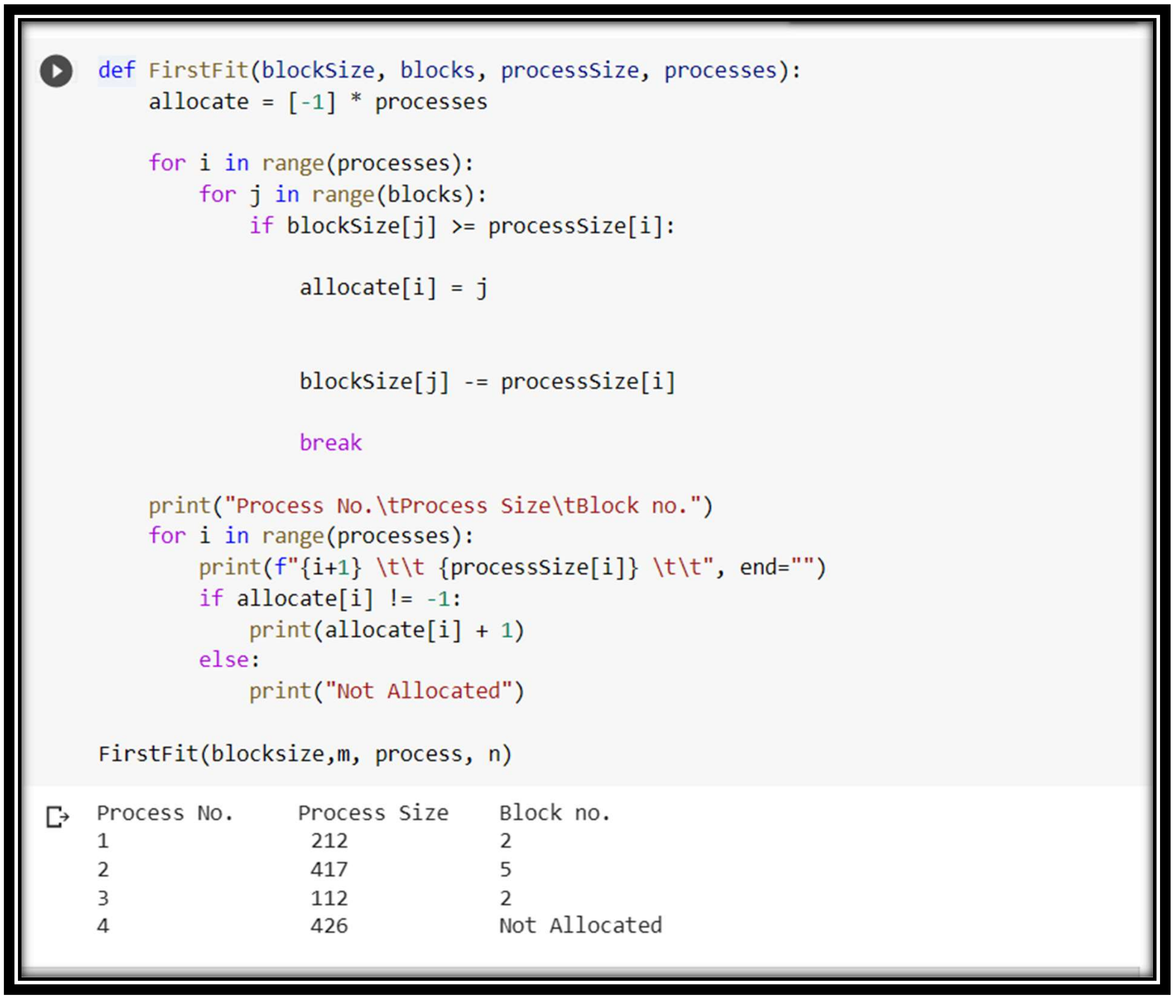
INPUTS



```
m = int(input("Enter number of fragments: "))
blocksize=[]
process=[]
index=[]
for i in range(m):
    x=int(input("Enter the size of bolck: "))
    blocksize.append(x)

n = int(input("Enter number of processes: "))
for i in range(n):
    y=int(input("Enter the size of process : "))
    process.append(y)

print("Block sizes:",blocksize)
print("Process sizes:",process)
```

```
Enter number of fragments: 5
Enter the size of bolck: 100
Enter the size of bolck: 500
Enter the size of bolck: 200
Enter the size of bolck: 300
Enter the size of bolck: 600
Enter number of processes: 4
Enter the size of process : 212
Enter the size of process : 417
Enter the size of process : 112
Enter the size of process : 426
Block sizes: [100, 500, 200, 300, 600]
Process sizes: [212, 417, 112, 426]
```

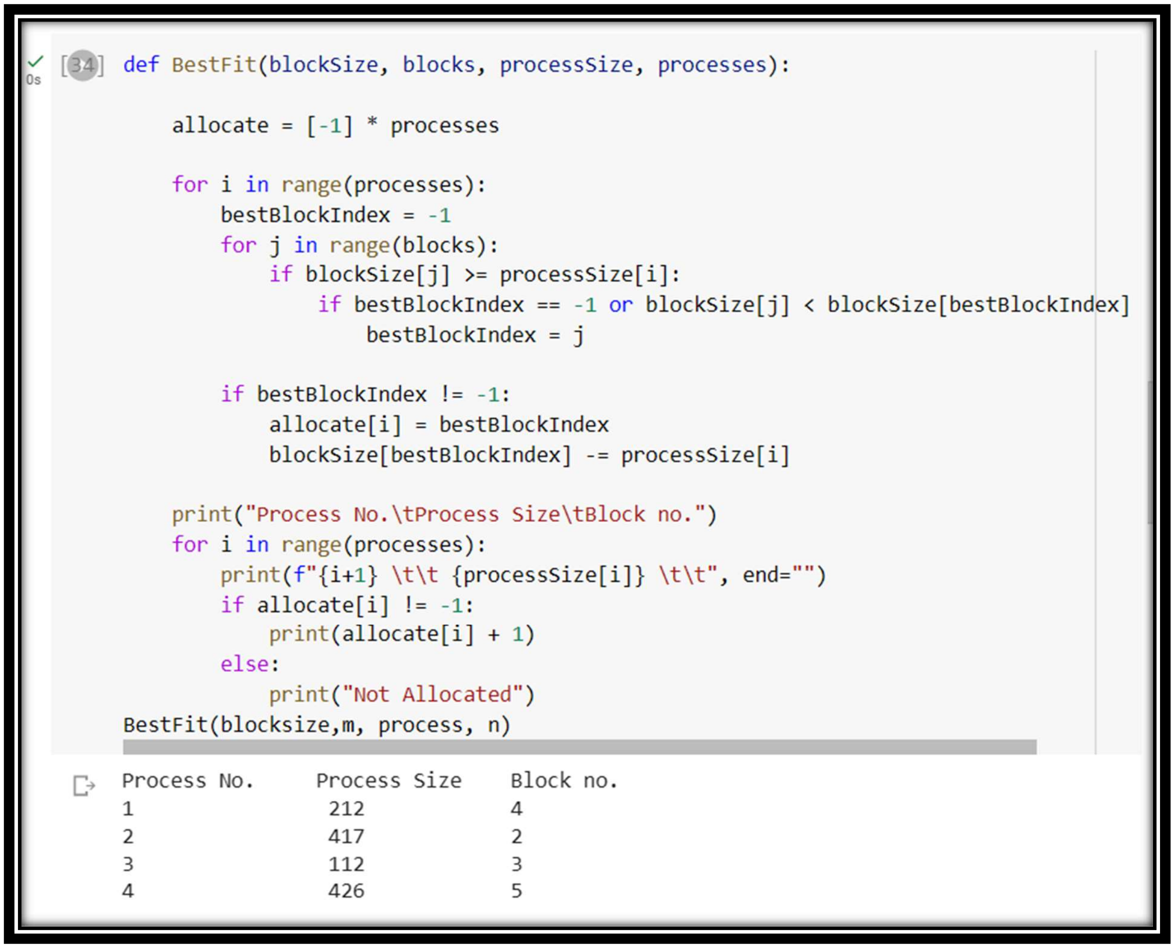
FIRST FIT



```
def FirstFit(blockSize, blocks, processSize, processes):
    allocate = [-1] * processes

    for i in range(processes):
        for j in range(blocks):
            if blockSize[j] >= processSize[i]:

                allocate[i] = j


                blockSize[j] -= processSize[i]

                break

    print("Process No.\tProcess Size\tBlock no.")
    for i in range(processes):
        print(f"{i+1} \t\t {processSize[i]} \t\t", end="")
        if allocate[i] != -1:
            print(allocate[i] + 1)
        else:
            print("Not Allocated")

FirstFit(blocksize,m, process, n)
```

```
Process No.     Process Size    Block no.
1                212            2
2                417            5
3                112            2
4                426            Not Allocated
```

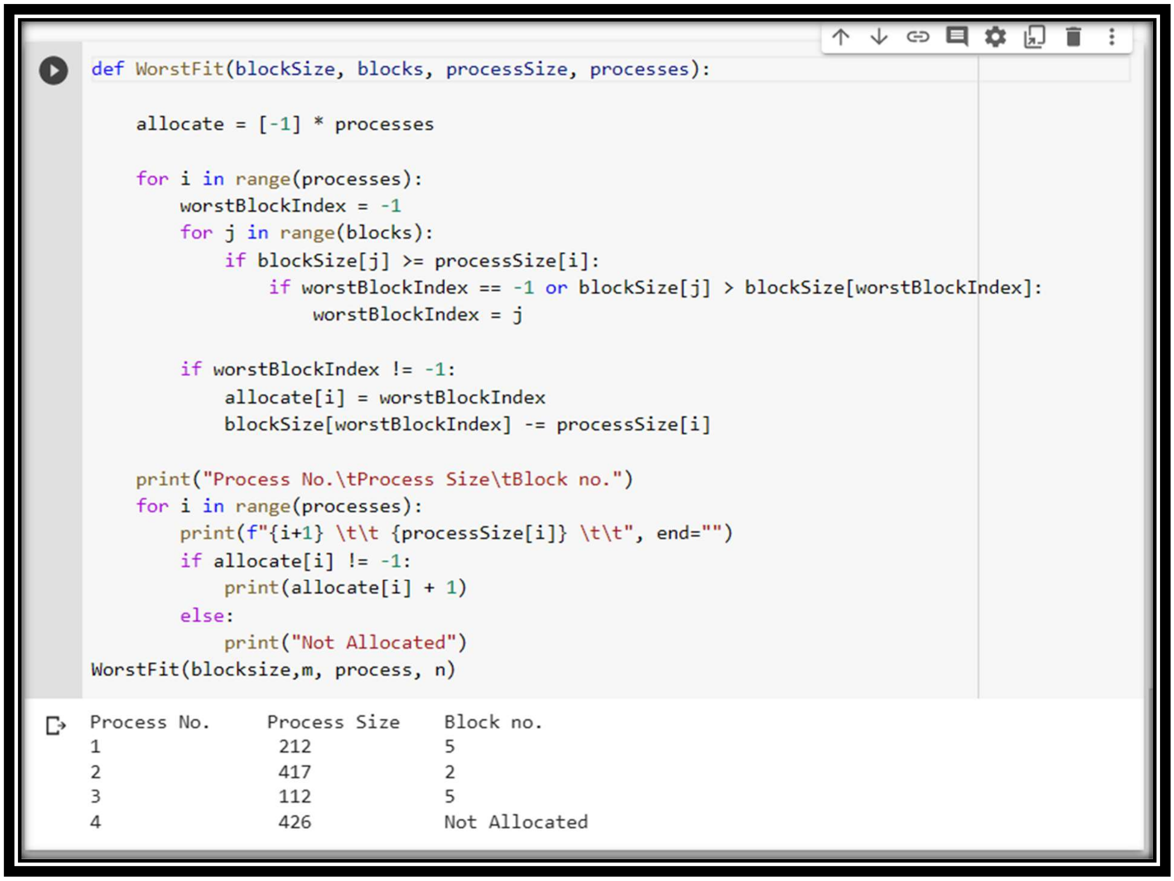
BEST FIT



```python
def BestFit(blockSize, blocks, processSize, processes):

    allocate = [-1] * processes

    for i in range(processes):
        bestBlockIndex = -1
        for j in range(blocks):
            if blockSize[j] >= processSize[i]:
                if bestBlockIndex == -1 or blockSize[j] < blockSize[bestBlockIndex]
                    bestBlockIndex = j

        if bestBlockIndex != -1:
            allocate[i] = bestBlockIndex
            blockSize[bestBlockIndex] -= processSize[i]

    print("Process No.\tProcess Size\tBlock no.")
    for i in range(processes):
        print(f"{i+1} \t\t {processSize[i]} \t\t", end="")
        if allocate[i] != -1:
            print(allocate[i] + 1)
        else:
            print("Not Allocated")
BestFit(blocksize,m, process, n)
```

```
Process No.     Process Size    Block no.
1                212            4
2                417            2
3                112            3
4                426            5
```

WORST FIT



```
def WorstFit(blockSize, blocks, processSize, processes):

    allocate = [-1] * processes

    for i in range(processes):
        worstBlockIndex = -1
        for j in range(blocks):
            if blockSize[j] >= processSize[i]:
                if worstBlockIndex == -1 or blockSize[j] > blockSize[worstBlockIndex]:
                    worstBlockIndex = j

        if worstBlockIndex != -1:
            allocate[i] = worstBlockIndex
            blockSize[worstBlockIndex] -= processSize[i]

    print("Process No.\tProcess Size\tBlock no.")
    for i in range(processes):
        print(f"{i+1} \t\t {processSize[i]} \t\t", end="")
        if allocate[i] != -1:
            print(allocate[i] + 1)
        else:
            print("Not Allocated")
WorstFit(blocksize,m, process, n)
```

```
Process No.     Process Size    Block no.
1                212            5
2                417            2
3                112            5
4                426            Not Allocated
```