Experiment 09

To write a program for the implementation of the First Fit & Best Fit algorithm

Learning Objective: Students should be able to understand First Fit & Best Fit algorithm by using different coding languages, such as C/C++, Java, and Python.

Tools: Online compiler

Theory:

First Fit Algorithm:

- Scans memory from the beginning.
- Allocates the first block that is large enough.
- Fast and simple to implement.
- Can lead to scattered small free memory spaces (fragmentation).
- Efficient in terms of speed but not always space utilization.

Best Fit Algorithm:

- Searches the entire memory list.
- Allocates the smallest block that is sufficient for the request.
- Aims to reduce wasted memory space.
- May take longer to find the best block.
- Can still cause fragmentation with small leftover spaces.

Code:

Best Fit -

```
g = 0; k = 0

class free:
        def __init__(self):
                self.tag=-1
                self.size=0
                self.next=None
free_head = None; prev_free = None

class alloc:
        def __init__(self):
                self.block_id=-1
                self.tag=-1
                self.size=0
                self.next=None

alloc_head = None;prev_alloc = None

def create_free(c):
```

```
        global g,prev_free,free_head
        p = free()
        p.size = c
        p.tag = g
        p.next = None
        if free_head is None:
                free_head = p
        else:
                prev_free.next = p
        prev_free = p
        g+=1

def print_free():
        p = free_head
        print("Tag\tSize")
        while (p != None) :
                print("{}\t{}".format(p.tag,p.size))
                p = p.next

def print_alloc():
        p = alloc_head
        print("Tag\tBlock ID\tSize")
        while (p is not None) :
                print("{}\t{}\t{}\t".format(p.tag,p.block_id,p.size))
                p = p.next

def create_alloc(c):
        global k,alloc_head

        q = alloc()
        q.size = c
        q.tag = k
        q.next = None
        p = free_head

        while (p != None) :
                if (q.size <= p.size):
                        break
                p = p.next

        if (p != None) :
                q.block_id = p.tag
                p.size -= q.size
                if (alloc_head == None):
                        alloc_head = q
                else :
```

```
print_alloc()
delete_alloc(0)
create_alloc(426)
print("After deleting block with tag id 0.")
print_alloc()
```

First Fit-

```
def first_fit(memory_blocks, process_sizes):
    allocation = [-1] * len(process_sizes)

    for i in range(len(process_sizes)):
        for j in range(len(memory_blocks)):
            if memory_blocks[j] >= process_sizes[i]:
                allocation[i] = j
                memory_blocks[j] -= process_sizes[i]
                break

    return allocation

# Example usage
memory_blocks = [100, 250, 200, 300, 150]
process_sizes = [150, 350, 200, 100]
allocation = first_fit(memory_blocks, process_sizes)
print("Memory Allocation:", allocation)
```

Output Screenshot:

```
                    Memory Management Scheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:4

Enter the size of the blocks:-
Block no.1:10
Block no.2:15
Block no.3:5
Block no.4:9
Block no.5:3
Enter the size of the processes :-
Process no.1:1
Process no.2:4
Process no.3:7
Process no.4:12

Process_no      Process_size    Block_no        Block_size      Fragment
1               1               5               3               2
2               4               3               5               1
3               7               4               9               2
4               12              2               15              3
Process returned 4 (0x4)   execution time : 33.196 s
Press any key to continue.
```

```
                        Memory Management Scheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:4

Enter the size of the blocks:-
Block no.1:10
Block no.2:15
Block no.3:5
Block no.4:9
Block no.5:3
Enter the size of the processes :-
Process no.1:1
Process no.2:4
Process no.3:7
Process no.4:12

Process_no      Process_size    Block_no        Block_size      Fragment
1               1               5               3               2
2               4               3               5               1
3               7               4               9               2
4               12              2               15              3
Process returned 4 (0x4)   execution time : 33.196 s
Press any key to continue.
```

Learning Outcomes: The student should have the ability to:

LO2.1 Outline various compilers for different language
LO2.2 Understood the First Fit / Best Fit Algorithm
LO2.3 Choose an appropriate compiler to solve the Memory allocation algorithm

Course Outcomes: Upon completion of the course, students will be able to learn about operating systems and security concepts.

Conclusion:

For Faculty Use:

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance/ Learning Attitude [20%]**	
Marks Obtained				