Set B Part 1

Design Marks: Total = 7

read_and_store(n, A, B)

// Array A and B of size *n* initialised into -1.

1. read the value of *m* //number of processes
2. initialise an array C of size *m* //to track the positions of processes in the order of *p_id*
3. j = 0 //keep track of index in array C
4. **for** i = 1 **to** *m*
   do read *p_id* //process id
   read *d* //duration
   compute $p = d^2 \bmod n$ //find position *p*
   do
   if A[*p*] = -1 //position *p* is vacant
   //assign *p_id, d into* position *p* of array A and B respectively
   then A[*p*] = *p_id*
   B[*p*] = *d*
   C[*j++*] = *p*
   else
   //find *k* which is the next vacant position
   for *pos* = 1 to *n*
   do
   compute $k = (p+pos) \bmod n$
   if A[k] = -1
   then
   A[*k*] = *p_id*
   B[*k*] = *d*
   C[*j++*] = *k*
   break // break out of for

5. **for** i = 1 **to** *m*
   print C[*i*] //print position *p* of each process in the order of *p_id*

Evaluation criteria : **[6 marks]**
Division:

- Read process id and duration of a process and store into the variables - 1 mark
- Find the position p of a process in the arrays A and B - 1 mark
- If position p is vacant, then store the details of the process in the arrays A and B at position p - 1 mark
- If position p is not vacant find next vacant position in the arrays A and B and store it- 1 mark
- Store the position p in an array as per the order of the process id and print the positions after storing m process details - 1 mark
- Correct function name, and number of arguments - 1 mark

list_process(n, A, B)

```
1. for i = 1 to n
       do
           if A[i] = -1
                   print -1  //array vacant
           else
                   print A[i]  B[i] separated by a space
```

Evaluation criteria : **[1 mark]**
Division: Print process id and duration of each process in the arrays separated by a space - 1 mark

Design Marks: Total = 3

Set B Part 2

read_and_store(n,m,A,B)

//A: 2D Array of size $n*m$ initialised into -1 (to store process id)

//B: 2D Array of size $n*m$ initialised into -1 (to store duration)

```
1. for j = 1 to m
        do
            read  p_id                  //process id
            read d                      //duration
            compute p = d^2 mod n       //find position p
            for  i = 1 to n
              do
                    if A[p][i] = -1
                      then
                          A[p][i]  = p_id
                          B[p][i]   = d
                          break   // break out of  for
```

Evaluation criteria : **[1 mark]** Division: • Read the process id and duration of a process and compute position, p - 0.25 mark • Selection of proper data structure to store the details of processes (if more than one process get same position p) - 0.75 mark

sort_process(n,m,A,B)

//A and B are 2-dimensional array of size $n*m$ with process id and duration respectively

```
1. for row = 1 to n
      do
          if  B[row][1]  !=  -1
            do
```

```
//apply any sorting algorithm on B[row], while swapping
elements in B[row] make changes accordingly in A[row]
//example given below uses Bubble sort algorithm
for i = 1 to m
    for  j = 1 to m-i
        do
            if B[row][j] > B[row][j+1]
                then
                    swap B[row][j] , B[row][j+1]
                    swap A[row][j] , A[row][j+1]
```

Evaluation criteria : **[1 mark]** Division: • Selection of a sorting algorithm - 0.25 mark • Sort array B (non-decreasing order of duration at position p), and reflect the same changes in array A - 0.75 mark

list_process(n,m,A,B)

//each new line prints process id and duration (separated by a comma) of the processes that are allotted to position p, where each process is separated by a space

```
1. for  i =1 to n
        j = 0
        do
            if A[i][j] = -1
                then
                    print('NULL')
            else
                while A[i][j] != -1
                    print A[i][j],B[i][j] print(' ')    //prints all the details of
                the process allotted to same position separated by space
                    j++     //increment j by one
```

Evaluation criteria : **[1 mark]** Division: Print the details of the processes in the given format - **1** mark