Set A Part 1

read_and_store(A, n)

```
//for each process, read its processing time and store it in the array A
for i = 1 to n
        read A[i]
```

Evaluation criteria : **[1 mark]**

Division: read processing time of each process and storing array A- 1 mark

run_processes(A, B, n, t)

```
p =  0      // keeps track of the number of processes completed
time =  0 //  keeps  track of the total time taken
while p < n
        for i = 1 to n
                if A[i]  > 0
                    if A[i]  > t
                        A[i] =  A[i] - t
                        time =  time + t
                    else if A[i] <= t
                        B[i] = time + A[i]        // set completion time in B[i]
                        A[i] = 0
                        time = time + t          // always increment by t
                        p =  p +1              // one more process completed
```

Evaluation criteria : **[5 marks]**

Division: Finding the next process to run - 2 marks

Calculating the time of completion - 3 marks

list_process(B, n)

```
// Prints the contents of the array B, with the elements separated by a single space
for i = 1 to n
    print B[i]; print('  ');
```

Evaluation criteria : **[1 mark]**

Division: print the time of completion of each process separated by a space - 1 mark

Set A Part 2

read(A, D, n)

```
1. read the value of n
2. for i ← 1 to n
        do  read A[i]     // arrival_time
            read D[i]     // processing time
```

Evaluation criteria : **[0.25 mark]**

Division: reading arrival time and processing time of each process and storing in array A and D respectively- 0.25 mark

run_processes(A, D, n, t)

//Runs all the processes by following the specifications given in the question.

```
1. time ← 0 //to track total time taken
2. while TRUE
        do ready_index ← -1      // index of the process that is selected for execution
           min_time ← 9999    // initialize to  a value greater than all possible
                              //  processing times
           for i ← 1 to n
               //select p_id s with arrival_time <= time and  p_time greater than 0
               do if A[i] <= time and D[i] > 0
                       //select p_id s with p_time less than min_time
                       then if D[i] < min_time
                               then min_time ← D[i]
                                    ready_index ← i
                       //select the p_id s with p_time equal to min_time
```

```
                else if D[i] = min_time
                        then if A[ready_index] > A[i]
                                then ready_index ← i
        if ready_index ≠ -1
                then if D[ready_index] > t
                        then D[ready_index] ← D[ready_index] - t
                             time ← time + t
                  else C[ready_index] ← time + D[ready_index]
                       time ← time + D[ready_index]
                       D[ready_index] ← 0
        else break  // break out of while
```

Evaluation criteria : **[2 marks]**

Division: finding the next process to run - 1 mark

Calculating the time of completion - 1 mark

```
list_process(C, n)
// Print the p_id and finishing time of each of the n processes
    //Arrange C[1 … n] in non-decreasing order using any sorting algorithm
    1.  for i ← 1 to n
                do for j ← n downto i + 1
                        do if C[j] < C[j - 1]
                        then exchange (C[j] , C[j - 1])
    2.  for i ← 1 to n
                do print i; print ‘  ‘; print C[i]; C
```

Evaluation criteria : **[0.75 mark]**

Division: sorting the process based on completion time - 0.75 mark