Ex: No: 7(i) **FCFS SCHEDULING**

AIM: To write a C program to implement FCFS scheduling algorithm.

ALGORITHM

1. Get the number of processes and their burst time.
2. Initialize the waiting time for process 1 and 0.
3. The waiting time for the other processes are calculated as follows:
   for(i=2;i<=n;i++),wt.p[i]=p[i-1]+bt.p[i-1].
4. The waiting time of all the processes is summed then average value time is calculated.
5. The waiting time of each process and average times are displayed.

PROGRAM

```
#include <stdio.h>
int main()
{
   int c = 0, i, n, bt[10], at[10], wt[10], ft[10];
   int st[10], tat[10];
   float awt = 0, atat = 0, rr[10];
   printf("Enter the number of process : ");
   scanf("%d", &n);
   for (i = 1; i <= n; i++)
   {
      printf("Enter the arrival time and burst time for the process %d : ", i);
      scanf("%d %d", &at[i], &bt[i]);
   }
   for (i = 1; i <= n; i++)
   {
      st[i] = c;
      c = c + bt[i];
      wt[i] = st[i] - at[i];
      ft[i] = st[i] + bt[i];
      tat[i] = wt[i] + bt[i];
      rr[i] = tat[i] / bt[i];
   }
   for (i = 1; i <= n; i++)
   {
      awt = awt + wt[i];
      atat = atat + tat[i];
   }
   awt = awt / n;
   atat = atat / n;
   printf("\n\t\t CPU SCHEDULING\n\t\t ****************");
   printf("\n\t\t FIRST COME FIRST SERVE\n\t\t **********************");
   printf("\n-------------------------------------------------------------\n");
   printf("proc\t at\t bt\t st\t ft\t wt\t tat\t rr\t\n");
```

```
    printf("--------------------------------------------------------------");
    for (i = 1; i <= n; i++)
    {
        printf("\n %d\t %d\t %d\t %d\t %d\t %d\t %d\t %5.2f", i, at[i], bt[i], st[i], ft[i], wt[i], tat[i],
rr[i]);
    }
    printf("\n--------------------------------------------------------------");
    printf("\n Average waiting time is %5.2f\n average tat is %5.2f\n\n", awt, atat);
}
```

Ex: No: 7(b) SJF SCHEDULING

AIM: To write a C program to implement Shortest Job First scheduling algorithm.

ALGORITHM

```
1. Start
2. Declare process variables and counters.
3.  Prompt for the number of processes (`n`).
4. Create an array of `Process` structures.
5. For each process (`i` from 1 to `n`), input ID, arrival time, and burst time.
6. Scheduling:
     a. While `completedCount` is less than `n`:
          i. Find the process with the smallest burst time that has arrived.
          ii. Update its start and finish times, waiting time, and turnaround time.
          iii. Increment `completedCount`.
7.  Display process details and calculate averages for waiting and turnaround times.
8. Stop.
```

PROGRAM

```
#include <stdio.h>

typedef struct {
    int processID;
    int arrivalTime;
    int burstTime;
    int startTime;
    int finishTime;
    int waitingTime;
    int turnaroundTime;
    float responseRatio;
} Process;

int main() {
    int n, nextStartTime = 0, completedCount = 0;
    float avgWaitingTime = 0, avgTurnaroundTime = 0;

    printf("\n\t SHORTEST JOB FIRST\n\t ******************");
    printf("\nEnter the number of processes to be executed: ");
    scanf("%d", &n);

    Process processes[n];

    printf("\nEnter process ID, arrival time, and burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d %d %d", &processes[i].processID, &processes[i].arrivalTime,
&processes[i].burstTime);
        processes[i].waitingTime = 0;
        processes[i].turnaroundTime = 0;
        processes[i].responseRatio = 0;
    }

    while (completedCount < n) {
        int minBurstIndex = -1, minBurst = 100;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrivalTime <= nextStartTime && processes[i].burstTime < minBurst &&
processes[i].burstTime > 0) {
                minBurst = processes[i].burstTime;
                minBurstIndex = i;
            }
        }

        if (minBurstIndex != -1) {
            int i = minBurstIndex;
            processes[i].startTime = nextStartTime;
            processes[i].finishTime = processes[i].startTime + processes[i].burstTime;
            processes[i].waitingTime = processes[i].startTime - processes[i].arrivalTime;
            processes[i].turnaroundTime = processes[i].waitingTime + processes[i].burstTime;
            processes[i].responseRatio = (float)processes[i].turnaroundTime / processes[i].burstTime;

            nextStartTime = processes[i].finishTime;
            processes[i].burstTime = 0;

            completedCount++;
        } else {
            nextStartTime++;
        }
    }

    printf("\n--------------------------------------");
    printf("\n PRO AT BT ST FT WT TT RR \n");
    printf("--------------------------------------\n");

    for (int i = 0; i < n; i++) {
        printf("%3d %2d %2d", processes[i].processID, processes[i].arrivalTime,
processes[i].burstTime);
        printf(" %3d %2d %2d %2d %4.2f\n", processes[i].startTime, processes[i].finishTime,
               processes[i].waitingTime, processes[i].turnaroundTime, processes[i].responseRatio);
        avgWaitingTime += processes[i].waitingTime;
        avgTurnaroundTime += processes[i].turnaroundTime;
    }

    avgWaitingTime /= n;
    avgTurnaroundTime /= n;

    printf("----------------------------------------\n");
    printf("Average waiting time: %5.2f\n", avgWaitingTime);
    printf("Average turnaround time: %5.2f\n", avgTurnaroundTime);

    return 0;
}
```