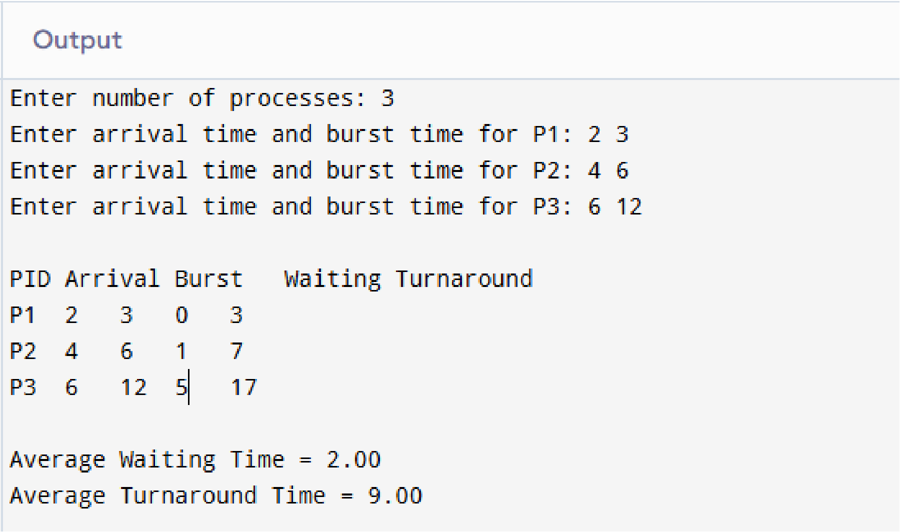
PROGRAM-3

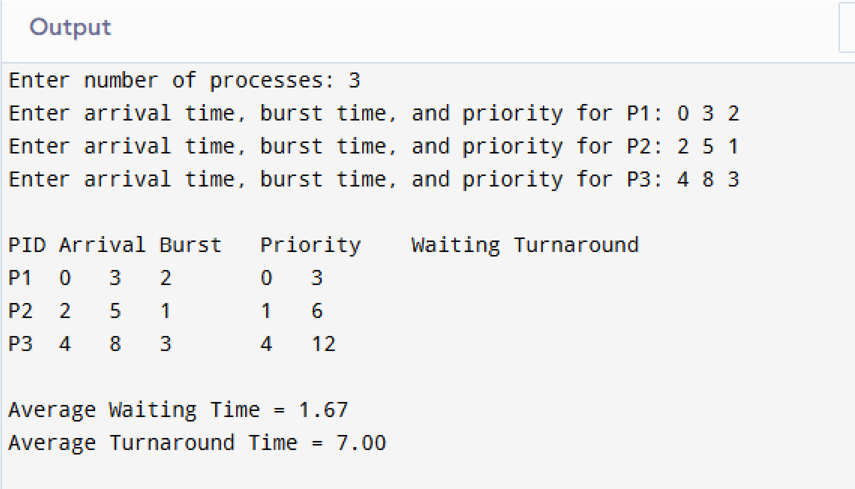
AIM- Write a program for shortest remaining time CPU scheduling.
CODE-

```
#include <stdio.h>
#define MAX 100

typedef struct {
    int pid;
    int arrival;
    int burst;
    int remaining;
    int waiting;
    int turnaround;
    int completed;
} Process;

void inputProcesses(Process p[], int n) {
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter arrival time and burst time for P%d: ", i + 1);
        scanf("%d %d", &p[i].arrival, &p[i].burst);
        p[i].remaining = p[i].burst;
        p[i].completed = 0;
    }
}

void SRTF(Process p[], int n) {
    int completed = 0, t = 0;
    float total_wait = 0, total_turnaround = 0;
    while (completed < n) {
        int min_index = -1;
        int min_remaining = 100000;
        for (int i = 0; i < n; i++) {
                if (!p[i].completed && p[i].arrival <= t && p[i].remaining < min_remaining &&
p[i].remaining > 0) {
                min_remaining = p[i].remaining;
                min_index = i;
            }
        }
        if (min_index == -1) {
            t++;
            continue;
        }

        p[min_index].remaining--;
        t++;
        if (p[min_index].remaining == 0) {
            p[min_index].completed = 1;
            completed++;

            p[min_index].turnaround = t - p[min_index].arrival;
            p[min_index].waiting = p[min_index].turnaround - p[min_index].burst;

            total_wait += p[min_index].waiting;
            total_turnaround += p[min_index].turnaround;
        }
    }
    printf("\nPID\tArrival\tBurst\tWaiting\tTurnaround\n");
    for (int i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].waiting, p[i].turnaround);
    printf("\nAverage Waiting Time = %.2f\n", total_wait / n);
    printf("Average Turnaround Time = %.2f\n", total_turnaround / n);
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    Process p[MAX];
    inputProcesses(p, n);
    SRTF(p, n);
    return 0;
}
```

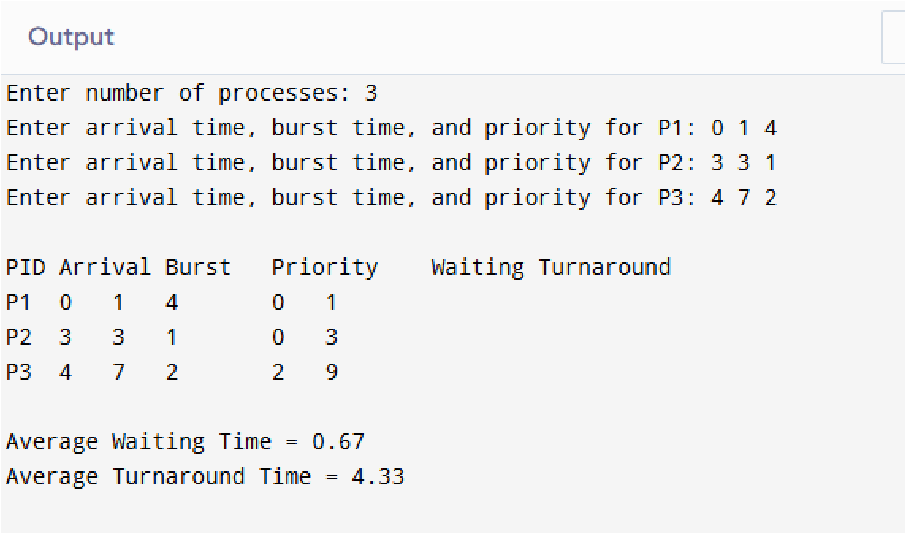


```
Enter number of processes: 3
Enter arrival time and burst time for P1: 2 3
Enter arrival time and burst time for P2: 4 6
Enter arrival time and burst time for P3: 6 12

PID Arrival Burst   Waiting Turnaround
P1   2    3    0    3
P2   4    6    1    7
P3   6    12   5    17

Average Waiting Time = 2.00
Average Turnaround Time = 9.00
```

Non premmptivw



```
Enter number of processes: 3
Enter arrival time, burst time, and priority for P1: 0 3 2
Enter arrival time, burst time, and priority for P2: 2 5 1
Enter arrival time, burst time, and priority for P3: 4 8 3

PID Arrival Burst   Priority    Waiting Turnaround
P1   0    3    2        0    3
P2   2    5    1        1    6
P3   4    8    3        4    12

Average Waiting Time = 1.67
Average Turnaround Time = 7.00
```



```
Enter number of processes: 3
Enter arrival time, burst time, and priority for P1: 0 1 4
Enter arrival time, burst time, and priority for P2: 3 3 1
Enter arrival time, burst time, and priority for P3: 4 7 2

PID Arrival Burst   Priority    Waiting Turnaround
P1   0    1    4        0    1
P2   3    3    1        0    3
P3   4    7    2        2    9

Average Waiting Time = 0.67
Average Turnaround Time = 4.33
```

Preemptive

PROGRAM-4

AIM- Write a program to perform priority scheduling.
CODE-

Non- Preemptive

```
#include <stdio.h>
typedef struct {
    int pid;
    int arrival;
    int burst;
    int priority;
    int waiting;
    int turnaround;
    int completed;
} Process;

void inputProcesses(Process p[], int n) {
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter arrival time, burst time, and priority for P%d: ", i + 1);
        scanf("%d %d %d", &p[i].arrival, &p[i].burst, &p[i].priority);
        p[i].completed = 0;
    }
}

void priorityScheduling(Process p[], int n) {
    int completed = 0, t = 0;
    float total_wait = 0, total_turnaround = 0;

    while (completed < n) {
        int idx = -1;
        int highest = 100000; // smaller number = higher priority

        for (int i = 0; i < n; i++) {
            if (!p[i].completed && p[i].arrival <= t && p[i].priority < highest) {
                highest = p[i].priority;
                idx = i;
            }
        }
        if (idx == -1) {
            t++;
            continue;
        }

        p[idx].waiting = t - p[idx].arrival;
        t += p[idx].burst;
        p[idx].turnaround = p[idx].waiting + p[idx].burst;
        p[idx].completed = 1;
        completed++;

        total_wait += p[idx].waiting;
        total_turnaround += p[idx].turnaround;
    }

    printf("\nPID\tArrival\tBurst\tPriority\tWaiting\tTurnaround\n");
    for (int i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\t\t%d\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].priority, p[i].waiting,
p[i].turnaround);

    printf("\nAverage Waiting Time = %.2f\n", total_wait / n);
    printf("Average Turnaround Time = %.2f\n", total_turnaround / n);
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    Process p[n];
    inputProcesses(p, n);
    priorityScheduling(p, n);

    return 0;
}
```

Preemptive

```
#include <stdio.h>
#define MAX 100

typedef struct {
    int pid;
    int arrival;
    int burst;
    int remaining;
    int priority;
    int waiting;
    int turnaround;
    int completed;
} Process;

// Input process details
void inputProcesses(Process p[], int n) {
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter arrival time, burst time, and priority for P%d: ", i + 1);
        scanf("%d %d %d", &p[i].arrival, &p[i].burst, &p[i].priority);
        p[i].remaining = p[i].burst;
        p[i].completed = 0;
    }
}

// Preemptive Priority Scheduling
void preemptivePriority(Process p[], int n) {
    int completed = 0, t = 0;
    float total_wait = 0, total_turnaround = 0;

    while (completed < n) {
        int idx = -1;
        int highest_priority = 100000; // lower number = higher priority

        for (int i = 0; i < n; i++) {
            if (!p[i].completed && p[i].arrival <= t && p[i].priority < highest_priority && p[i].remaining >
0) {
                highest_priority = p[i].priority;
                idx = i;
            }
        }
        if (idx == -1) {
            t++; // CPU idle
            continue;
        }

        // Execute one time unit
        p[idx].remaining--;
        t++;

        // If process finished
        if (p[idx].remaining == 0) {
            p[idx].completed = 1;
            completed++;

            p[idx].turnaround = t - p[idx].arrival;
            p[idx].waiting = p[idx].turnaround - p[idx].burst;

            total_wait += p[idx].waiting;
            total_turnaround += p[idx].turnaround;
        }
    }

    printf("\nPID\tArrival\tBurst\tPriority\tWaiting\tTurnaround\n");
    for (int i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\t\t%d\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].priority, p[i].waiting,
p[i].turnaround);

    printf("\nAverage Waiting Time = %.2f\n", total_wait / n);
    printf("Average Turnaround Time = %.2f\n", total_turnaround / n);
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    Process p[MAX];
    inputProcesses(p, n);
    preemptivePriority(p, n);
    return 0;
}
```