## 2. Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.

```
#include <stdio.h>
#include <limits.h>

// Define the number of processes
#define N 5

// Structure to store process information
typedef struct {
   int id;        // Process ID
   int burst;     // Burst time
   int arrival;   // Arrival time
   int priority;  // Priority (for Priority Scheduling)
   int waiting;   // Waiting time
   int turnaround; // Turnaround time
} Process;

// Function prototypes
void FCFS(Process processes[], int n);
void SJF(Process processes[], int n);
void RoundRobin(Process processes[], int n, int quantum);
void PriorityScheduling(Process processes[], int n);

int main() {
   Process processes[N] = {
      {1, 6, 0, 2, 0, 0},
      {2, 8, 1, 1, 0, 0},
      {3, 7, 2, 3, 0, 0},
      {4, 3, 3, 4, 0, 0},
      {5, 4, 4, 5, 0, 0}
   };

   printf("FCFS Scheduling:\n");
   FCFS(processes, N);
```

```
    printf("\nSJF Scheduling:\n");
    SJF(processes, N);

    printf("\nRound Robin Scheduling:\n");
    RoundRobin(processes, N, 4); // Time quantum is 4

    printf("\nPriority Scheduling:\n");
    PriorityScheduling(processes, N);

    return 0;
}

void FCFS(Process processes[], int n) {
    int total_wt = 0, total_tt = 0;
    int current_time = 0;

    for (int i = 0; i < n; i++) {
        if (current_time < processes[i].arrival)
            current_time = processes[i].arrival;

        processes[i].waiting = current_time - processes[i].arrival;
        processes[i].turnaround = processes[i].waiting + processes[i].burst;
        current_time += processes[i].burst;

        total_wt += processes[i].waiting;
        total_tt += processes[i].turnaround;

        printf("Process %d: Waiting Time = %d, Turnaround Time = %d\n",
               processes[i].id, processes[i].waiting, processes[i].turnaround);
    }

    printf("Average Waiting Time = %.2f\n", (float)total_wt / n);
    printf("Average Turnaround Time = %.2f\n", (float)total_tt / n);
}

void SJF(Process processes[], int n) {
```

```
    int total_wt = 0, total_tt = 0;
    int current_time = 0;
    int completed = 0;
    Process temp;

    // Sort processes by burst time
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (processes[i].burst > processes[j].burst) {
                temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }

    for (int i = 0; i < n; i++) {
        if (current_time < processes[i].arrival)
            current_time = processes[i].arrival;

        processes[i].waiting = current_time - processes[i].arrival;
        processes[i].turnaround = processes[i].waiting + processes[i].burst;
        current_time += processes[i].burst;

        total_wt += processes[i].waiting;
        total_tt += processes[i].turnaround;

        printf("Process %d: Waiting Time = %d, Turnaround Time = %d\n",
               processes[i].id, processes[i].waiting, processes[i].turnaround);
    }

    printf("Average Waiting Time = %.2f\n", (float)total_wt / n);
    printf("Average Turnaround Time = %.2f\n", (float)total_tt / n);
}

void RoundRobin(Process processes[], int n, int quantum) {
    int remaining_burst[N];
```

```
    int total_wt = 0, total_tt = 0;
    int current_time = 0;
    int completed = 0;

    for (int i = 0; i < n; i++) {
        remaining_burst[i] = processes[i].burst;
    }

    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (remaining_burst[i] > 0) {
                if (remaining_burst[i] > quantum) {
                    remaining_burst[i] -= quantum;
                    current_time += quantum;
                } else {
                    current_time += remaining_burst[i];
                    processes[i].waiting = current_time - processes[i].arrival -
processes[i].burst;
                    processes[i].turnaround = current_time - processes[i].arrival;
                    total_wt += processes[i].waiting;
                    total_tt += processes[i].turnaround;
                    remaining_burst[i] = 0;
                    completed++;
                }
            }
        }
    }

    for (int i = 0; i < n; i++) {
        printf("Process %d: Waiting Time = %d, Turnaround Time = %d\n",
                processes[i].id, processes[i].waiting, processes[i].turnaround);
    }

    printf("Average Waiting Time = %.2f\n", (float)total_wt / n);
    printf("Average Turnaround Time = %.2f\n", (float)total_tt / n);
}
```

```
void PriorityScheduling(Process processes[], int n) {
   int total_wt = 0, total_tt = 0;
   int current_time = 0;
   Process temp;

   // Sort processes by priority (highest priority first)
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (processes[i].priority > processes[j].priority) {
            temp = processes[i];
            processes[i] = processes[j];
            processes[j] = temp;
         }
      }
   }

   for (int i = 0; i < n; i++) {
      if (current_time < processes[i].arrival)
         current_time = processes[i].arrival;

      processes[i].waiting = current_time - processes[i].arrival;
      processes[i].turnaround = processes[i].waiting + processes[i].burst;
      current_time += processes[i].burst;

      total_wt += processes[i].waiting;
      total_tt += processes[i].turnaround;

      printf("Process %d: Waiting Time = %d, Turnaround Time = %d\n",
            processes[i].id, processes[i].waiting, processes[i].turnaround);
   }

   printf("Average Waiting Time = %.2f\n", (float)total_wt / n);
   printf("Average Turnaround Time = %.2f\n", (float)total_tt / n);
}
```