Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time. (Any one) a) FCFS b) SJF c) Priority d) Round Robin (Experiment with different quantum sizes for RR algorithm)

```
#include <stdio.h>

void sort(int proc_id[], int at[], int bt[], int n) {
    int min, temp;
    for (int i = 0; i < n; i++) {
        min = at[i];
        for (int j = i; j < n; j++) {
            if (at[j] < min) {
                temp = at[i];
                at[i] = at[j];
                at[j] = temp;

                temp = bt[j];
                bt[j] = bt[i];
                bt[i] = temp;

                temp = proc_id[i];
                proc_id[i] = proc_id[j];
                proc_id[j] = temp;
            }
        }
    }
}

int main() {
    int n, c = 0;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    int proc_id[n], at[n], bt[n], ct[n], tat[n], wt[n];


    for (int i = 0; i < n; i++) {
        proc_id[i] = i + 1;
    }

    printf("Enter arrival times:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &at[i]);
    }
```

```
    printf("Enter burst times:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &bt[i]);
    }

    sort(proc_id, at, bt, n);

    for (int i = 0; i < n; i++) {
        if (c >= at[i])
            c += bt[i];
        else
            c += at[i]  + bt[i];
        ct[i] = c;
    }

    for (int i = 0; i < n; i++) {
        tat[i] = ct[i] - at[i];
    }


    for (int i = 0; i < n; i++) {
        wt[i] = tat[i] - bt[i];
    }

    printf("FCFS scheduling:\n");
    printf("PID\tAT\tBT\tCT\tTAT\tWT\n");

    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", proc_id[i], at[i], bt[i], ct[i], tat[i], wt[i]);
    }


    return 0;
}
```

Output

```
Enter the number of processes: 4
Enter the arrival times of the processes: 0 1 2 3
Enter the burst times of the processes: 10 5 8 12

Process  Arrival Time  Burst Time  Completion Time  Turnaround Time  Waiting Time
P1       0             10          10               10               0
P2       1             5           15               14               9
P3       2             8           23               21               13
P4       3             12          35               32               20

Average Turnaround Time: 14.25
Average Waiting Time: 10.5
```

2.SJF

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

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void sortByArrivalAndBurst(int n, int pid[], int at[], int bt[]) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (at[j] > at[j + 1] || (at[j] == at[j + 1] && bt[j] > bt[j + 1])) {
                swap(&pid[j], &pid[j + 1]);
                swap(&at[j], &at[j + 1]);
                swap(&bt[j], &bt[j + 1]);
            }
        }
    }
}

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

    int pid[n], at[n], bt[n], wt[n], tat[n], ct[n];
    bool completed[n];
```

```
    for (int i = 0; i < n; i++) {
        pid[i] = i + 1;
        printf("Enter arrival time for process %d: ", i + 1);
        scanf("%d", &at[i]);
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &bt[i]);
        completed[i] = false;
    }

    sortByArrivalAndBurst(n, pid, at, bt);

    int currentTime = 0, completedProcesses = 0;

    while (completedProcesses < n) {
        int shortestIndex = -1;
        int minBurst = 10000;

        for (int i = 0; i < n; i++) {
            if (!completed[i] && at[i] <= currentTime && bt[i] < minBurst) {
                minBurst = bt[i];
                shortestIndex = i;
            }
        }

        if (shortestIndex == -1) {
            currentTime++;
        } else {
            ct[shortestIndex] = currentTime + bt[shortestIndex];
            tat[shortestIndex] = ct[shortestIndex] - at[shortestIndex];
            wt[shortestIndex] = tat[shortestIndex] - bt[shortestIndex];
            currentTime = ct[shortestIndex];
            completed[shortestIndex] = true;
            completedProcesses++;
        }
    }

    printf("\nProcess ID\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround
Time\n");
    float avg_wt = 0, avg_tat = 0;
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", pid[i], at[i], bt[i], ct[i], wt[i], tat[i]);
        avg_wt += wt[i];
        avg_tat += tat[i];
```

```
    }
    printf("\nAverage Waiting Time: %.2f", avg_wt / n);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat / n);

    return 0;
}
```

```
Enter the number of processes: 4
Enter the arrival times of the processes: 0 1 2 3
Enter the burst times of the processes: 10 5 8 12

Process  Arrival Time  Burst Time  Completion Time  Turnaround Time  Waiting Time
P1       0             10          10               10               0
P2       1             5           15               14               9
P3       2             8           23               21               13
P4       3             12          35               32               20

Average Turnaround Time: 14.25
Average Waiting Time: 10.5
```

3.Round robin

```
#include <stdio.h>

void findWaitingTime(int n, int bt[], int wt[], int quantum) {
    int rem_bt[n];
    for (int i = 0; i < n; i++) {
        rem_bt[i] = bt[i];
    }

    int time = 0;
    int completed = 0;
    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (rem_bt[i] > 0) {
                if (rem_bt[i] <= quantum) {
                    time += rem_bt[i];
                    wt[i] = time - bt[i];
                    rem_bt[i] = 0;
                    completed++;
                } else {
                    rem_bt[i] -= quantum;
                    time += quantum;
                }
```

```
            }
        }
    }
}

void findTurnaroundTime(int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
    }
}

void findAverageTime(int n, int bt[], int quantum) {
    int wt[n], tat[n];
    findWaitingTime(n, bt, wt, quantum);
    findTurnaroundTime(n, bt, wt, tat);

    int total_wt = 0, total_tat = 0;
    printf("\nProcess ID\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
        printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f", (float)total_wt / n);
    printf("\nAverage Turnaround Time: %.2f", (float)total_tat / n);
}

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

    int bt[n];
    printf("Enter the burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Burst time of process %d: ", i + 1);
        scanf("%d", &bt[i]);
    }

    printf("Enter the time quantum: ");
    scanf("%d", &quantum);

    findAverageTime(n, bt, quantum);
```

```
    return 0;
}
```

```
Enter the number of processes: 4
Enter the arrival times of the processes: 0 1 2 3
Enter the burst times of the processes: 10 5 8 12
Enter the quantum size: 3

Process  Arrival Time  Burst Time  Completion Time  Turnaround Time  Waiting Time
P1       0             10          18               18               8
P2       1             5           8                7                2
P3       2             8           21               19               11
P4       3             12          30               27               15

Average Turnaround Time: 17.75
Average Waiting Time: 9
```

4.Priority

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

void sortProcessesByArrivalTime(int n, int arrival[], int burst[], int priority[], int pid[]) {
    int temp, i, j;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arrival[j] > arrival[j + 1] || (arrival[j] == arrival[j + 1] && priority[j] > priority[j + 1])) {
                // Swap the processes based on arrival time and priority
                temp = arrival[j]; arrival[j] = arrival[j + 1]; arrival[j + 1] = temp;
                temp = burst[j]; burst[j] = burst[j + 1]; burst[j + 1] = temp;
                temp = priority[j]; priority[j] = priority[j + 1]; priority[j + 1] = temp;
                temp = pid[j]; pid[j] = pid[j + 1]; pid[j + 1] = temp;
            }
        }
    }
}

void preemptivePriorityScheduling(int n, int arrival[], int burst[], int priority[], int pid[]) {
    int remainingBurst[n], completionTime[n], waitingTime[n], turnaroundTime[n];
    bool completed[n];
    int currentTime = 0, completedProcesses = 0, shortestIndex;
    for (int i = 0; i < n; i++) {
        remainingBurst[i] = burst[i];
```

```
        completed[i] = false;
    }

    while (completedProcesses < n) {
        int minPriority = 10000;
        shortestIndex = -1;

        for (int i = 0; i < n; i++) {
            if (!completed[i] && arrival[i] <= currentTime && priority[i] < minPriority) {
                minPriority = priority[i];
                shortestIndex = i;
            }
        }

        if (shortestIndex != -1) {
            remainingBurst[shortestIndex]--;
            if (remainingBurst[shortestIndex] == 0) {
                completed[shortestIndex] = true;
                completedProcesses++;
                completionTime[shortestIndex] = currentTime + 1;
                turnaroundTime[shortestIndex] = completionTime[shortestIndex] -
arrival[shortestIndex];
                waitingTime[shortestIndex] = turnaroundTime[shortestIndex] - burst[shortestIndex];
            }
        }

        currentTime++;
    }

    printf("\nProcess ID\tArrival Time\tBurst Time\tPriority\tCompletion Time\tWaiting
Time\tTurnaround Time\n");
    float totalWaitingTime = 0, totalTurnaroundTime = 0;
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", pid[i], arrival[i], burst[i], priority[i],
completionTime[i], waitingTime[i], turnaroundTime[i]);
        totalWaitingTime += waitingTime[i];
        totalTurnaroundTime += turnaroundTime[i];
    }

    printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
    printf("Average Turnaround Time: %.2f\n", totalTurnaroundTime / n);
}

void nonPreemptivePriorityScheduling(int n, int arrival[], int burst[], int priority[], int pid[]) {
```

```
    int completionTime[n], waitingTime[n], turnaroundTime[n];
    bool completed[n];
    int currentTime = 0, completedProcesses = 0;
    for (int i = 0; i < n; i++) {
        completed[i] = false;
    }

    while (completedProcesses < n) {
        int highestPriority = 10000, selectedProcess = -1;
        for (int i = 0; i < n; i++) {
            if (!completed[i] && arrival[i] <= currentTime && priority[i] < highestPriority) {
                highestPriority = priority[i];
                selectedProcess = i;
            }
        }

        if (selectedProcess != -1) {
            currentTime += burst[selectedProcess];
            completionTime[selectedProcess] = currentTime;
            turnaroundTime[selectedProcess] = completionTime[selectedProcess] -
arrival[selectedProcess];
            waitingTime[selectedProcess] = turnaroundTime[selectedProcess] -
burst[selectedProcess];
            completed[selectedProcess] = true;
            completedProcesses++;
        } else {
            currentTime++;
        }
    }

    printf("\nProcess ID\tArrival Time\tBurst Time\tPriority\tCompletion Time\tWaiting
Time\tTurnaround Time\n");
    float totalWaitingTime = 0, totalTurnaroundTime = 0;
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", pid[i], arrival[i], burst[i], priority[i],
completionTime[i], waitingTime[i], turnaroundTime[i]);
        totalWaitingTime += waitingTime[i];
        totalTurnaroundTime += turnaroundTime[i];
    }

    printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
    printf("Average Turnaround Time: %.2f\n", totalTurnaroundTime / n);
}
```

```
int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int arrival[n], burst[n], priority[n], pid[n];

    for (int i = 0; i < n; i++) {
        pid[i] = i + 1;
        printf("Enter arrival time, burst time, and priority for process %d: ", i + 1);
        scanf("%d%d%d", &arrival[i], &burst[i], &priority[i]);
    }

    sortProcessesByArrivalTime(n, arrival, burst, priority, pid);

    printf("Preemptive Priority Scheduling:\n");
    preemptivePriorityScheduling(n, arrival, burst, priority, pid);

    printf("\nNon-Preemptive Priority Scheduling:\n");
    nonPreemptivePriorityScheduling(n, arrival, burst, priority, pid);

    return 0;
```

```
Enter the number of processes: 4
Enter the arrival times of the processes: 0 1 2 3
Enter the burst times of the processes: 10 5 8 12
Enter the priorities of the processes: 2 1 4 3

Process  Arrival Time  Burst Time  Completion Time  Turnaround Time  Waiting Time
P1       0             10          10               10               0
P2       1             5           15               14               9
P3       2             8           23               21               13
P4       3             12          35               32               20

Average Turnaround Time: 14.25
Average Waiting Time: 10.5
```