Q-Write a program to implement Shortest Job First (SJF) job scheduling algorithm.

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
struct Process {
    int id;
    int arrivalTime;
    int burstTime;
};
bool compareBurstTime(const Process& p1, const Process& p2) {
    return p1.burstTime < p2.burstTime;
}

void sjfScheduling(vector<Process>& processes) {
    sort(processes.begin(), processes.end(), compareBurstTime);

    int currentTime = 0;
    double totalWaitingTime = 0;
    double totalTurnaroundTime = 0;

    for (int i = 0; i < processes.size(); ++i) {
        cout << "Process " << processes[i].id << ": Waiting Time = " << currentTime << ", Turnaround Time = " << currentTime + processes[i].burstTime << endl;
        totalWaitingTime += currentTime;
        totalTurnaroundTime += currentTime + processes[i].burstTime;
        currentTime += processes[i].burstTime;
    }
```

```
    double avgWaitingTime = totalWaitingTime / processes.size();
    double avgTurnaroundTime = totalTurnaroundTime / processes.size();

    cout << "Average Waiting Time = " << avgWaitingTime << endl;
    cout << "Average Turnaround Time = " << avgTurnaroundTime << endl;
}

int main() {
    vector<Process> processes = {
        {1, 0, 5},
        {2, 1, 3},
        {3, 2, 8},
        {4, 3, 6}
    };

    sjfScheduling(processes);

    return 0;
}
```

Q-Write a program to implement Shortest Remaining Time First (SRTF) job scheduling algorithm.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Process {
    int id;
    int arrivalTime;
```

```
    int burstTime;
    int remainingTime;
};

bool compareArrivalTime(const Process& p1, const Process& p2) {
    return p1.arrivalTime < p2.arrivalTime;
}

bool compareRemainingTime(const Process& p1, const Process& p2) {
    return p1.remainingTime < p2.remainingTime;
}

void srtfScheduling(vector<Process>& processes) {
    sort(processes.begin(), processes.end(), compareArrivalTime);

    int currentTime = 0;
    int completedProcesses = 0;

    while (completedProcesses < processes.size()) {
        int shortestIndex = -1;
        int shortestTime = INT_MAX;

        for (int i = 0; i < processes.size(); ++i) {
            if (processes[i].arrivalTime <= currentTime && processes[i].remainingTime > 0) {
                if (processes[i].remainingTime < shortestTime) {
                    shortestIndex = i;
                    shortestTime = processes[i].remainingTime;
                }
            }
        }
```

```
        if (shortestIndex == -1) {
            currentTime++;
            continue;
        }

        processes[shortestIndex].remainingTime--;
        currentTime++;

        if (processes[shortestIndex].remainingTime == 0) {
            cout << "Process " << processes[shortestIndex].id << ": Turnaround Time = " << currentTime - processes[shortestIndex].arrivalTime << endl;
            completedProcesses++;
        }
    }
}

int main() {
    vector<Process> processes = {
        {1, 0, 5, 5},
        {2, 1, 3, 3},
        {3, 2, 8, 8},
        {4, 3, 6, 6}
    };

    srtfScheduling(processes);

    return 0;
}
```