SOLUTION:3 PREEMPTIVE SJF WITHOUT PROCESS STRUCTURE CONCEPT

```c
#include <stdio.h>

int main() {
    int n, time = 0, remain, smallest = 0, endTime;
    int arrival[10], burst[10], remaining[10], waiting[10], turnaround[10], finished[10];
    float avgWaiting = 0, avgTurnaround = 0;

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

    remain = n; // Remaining processes

    // Input arrival and burst times
    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d Arrival Time: ", i + 1);
        scanf("%d", &arrival[i]);
        printf("Process %d Burst Time: ", i + 1);
        scanf("%d", &burst[i]);
        remaining[i] = burst[i];  // Initialize remaining time as burst time
        finished[i] = 0;          // Process is not finished yet
    }

    // Set a high value to help find the smallest remaining time
    remaining[9] = 9999;

    printf("\nGantt Chart:\nTime\tProcess\n");

    // Loop until all processes are completed
```

```
for (time = 0; remain != 0; time++) {
    smallest = 9; // Assuming an invalid process initially

    // Find the process with the smallest remaining burst time that has arrived
    for (int i = 0; i < n; i++) {
        if (arrival[i] <= time && remaining[i] > 0 && remaining[i] < remaining[smallest]) {
            smallest = i;
        }
    }

    // If no valid process is found, continue
    if (smallest == 9) {
        continue;
    }

    printf("%d\tP%d\n", time, smallest + 1);
    remaining[smallest]--;  // Decrease the remaining time

    // If the process is finished
    if (remaining[smallest] == 0) {
        remain--;  // One less process to finish
        endTime = time + 1;  // The time when the process finished
        waiting[smallest] = endTime - arrival[smallest] - burst[smallest];  // Calculate waiting time
        turnaround[smallest] = endTime - arrival[smallest];  // Turnaround time
        avgWaiting += waiting[smallest];
        avgTurnaround += turnaround[smallest];
    }
}

// Output results
printf("\nProcess\tArrival\tBurst\tWaiting\tTurnaround\n");
```

```
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\n", i + 1, arrival[i], burst[i], waiting[i], turnaround[i]);
    }

    // Display average waiting time and turnaround time
    avgWaiting /= n;
    avgTurnaround /= n;
    printf("\nAverage Waiting Time: %.2f\n", avgWaiting);
    printf("Average Turnaround Time: %.2f\n", avgTurnaround);

    return 0;
}
```