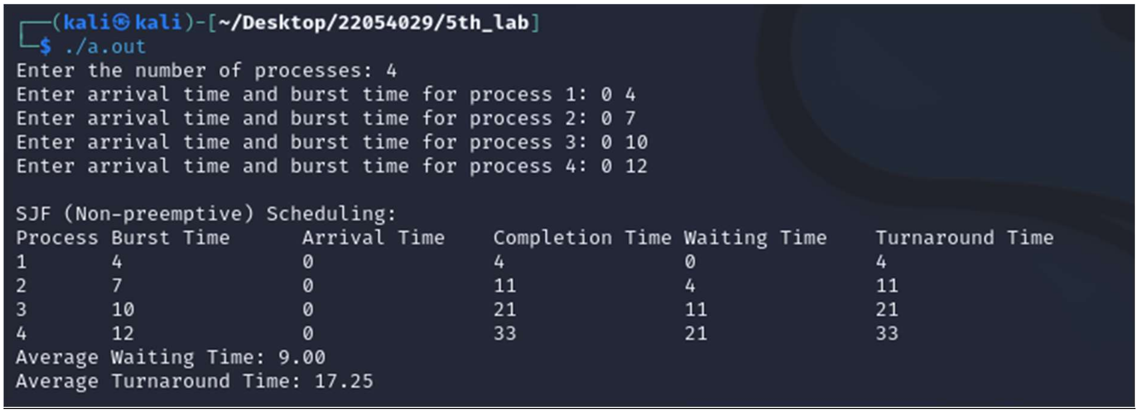
1) SJF(non-preemptive)

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

// Process structure
struct Process {
    int pid;  // Process ID
    int burst_time;  // Burst time
    int arrival_time; // Arrival time
};

// Function to perform SJF scheduling (non-preemptive)
void sjf_non_preemptive(struct Process *processes, int n) {
    int total_waiting_time = 0, total_turnaround_time = 0;
    int completion_time[n], turnaround_time[n], waiting_time[n];

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

    completion_time[0] = processes[0].burst_time;
    turnaround_time[0] = completion_time[0] - processes[0].arrival_time;
    waiting_time[0] = turnaround_time[0] - processes[0].burst_time;

    total_waiting_time += waiting_time[0];
    total_turnaround_time += turnaround_time[0];

    for (int i = 1; i < n; i++) {
        completion_time[i] = completion_time[i - 1] + processes[i].burst_time;
        turnaround_time[i] = completion_time[i] - processes[i].arrival_time;
        waiting_time[i] = turnaround_time[i] - processes[i].burst_time;

        total_waiting_time += waiting_time[i];
        total_turnaround_time += turnaround_time[i];
    }

    // Print result
    printf("Process\tBurst Time\tArrival Time\tCompletion Time\tWaiting Time\tTurnaround
Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].burst_time,
              processes[i].arrival_time, completion_time[i], waiting_time[i], turnaround_time[i]);
    }
    // Print average waiting time and average turnaround time
    printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
}

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

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

    printf("\nSJF (Non-preemptive) Scheduling:\n");
    sjf_non_preemptive(processes, n);

    return 0;
}
```

Output



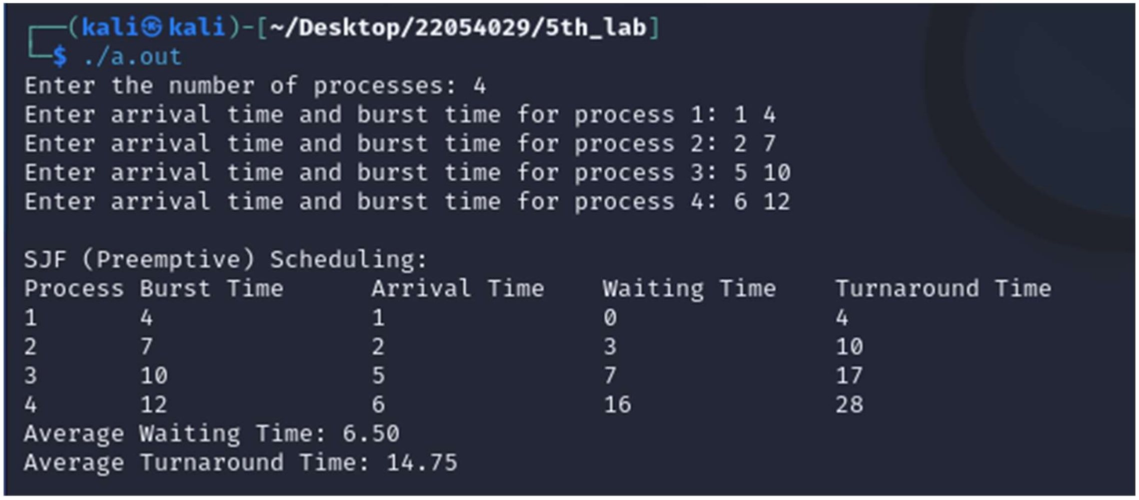
```
┌──(kali㉿kali)-[~/Desktop/22054029/5th_lab]
└─$ ./a.out
Enter the number of processes: 4
Enter arrival time and burst time for process 1: 0 4
Enter arrival time and burst time for process 2: 0 7
Enter arrival time and burst time for process 3: 0 10
Enter arrival time and burst time for process 4: 0 12

SJF (Non-preemptive) Scheduling:
Process Burst Time      Arrival Time    Completion Time Waiting Time    Turnaround Time
1       4               0               4               0               4
2       7               0               11              4               11
3       10              0               21              11              21
4       12              0               33              21              33
Average Waiting Time: 9.00
Average Turnaround Time: 17.25
```

2) SJF (preemptive)

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

// Process structure
struct Process {
    int pid;  // Process ID
    int burst_time;  // Burst time
    int remaining_time; // Remaining burst time
    int arrival_time; // Arrival time
```

```
};

// Function to perform Preemptive SJF scheduling
void sjf_preemptive(struct Process *processes, int n) {
    int current_time = 0, completed = 0;
    int total_waiting_time = 0, total_turnaround_time = 0;
    int waiting_time[n], turnaround_time[n];

    // Initialize remaining time and waiting time
    for (int i = 0; i < n; i++) {
        processes[i].remaining_time = processes[i].burst_time;
        waiting_time[i] = 0;
        turnaround_time[i] = 0;
    }

    // Schedule processes until all are completed
    while (completed != n) {
        int min_remaining_time = INT_MAX;
        int min_remaining_time_index = -1;

        // Find process with minimum remaining time
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= current_time && processes[i].remaining_time <
min_remaining_time && processes[i].remaining_time > 0) {
                min_remaining_time = processes[i].remaining_time;
                min_remaining_time_index = i;
            }
        }

        // If no process is found, increment current time
        if (min_remaining_time_index == -1) {
            current_time++;
            continue;
        }

        // Execute the process for 1 unit of time
        processes[min_remaining_time_index].remaining_time--;
        current_time++;

        // If a process is completed
        if (processes[min_remaining_time_index].remaining_time == 0) {
            completed++;
            int completion_time = current_time;
            waiting_time[min_remaining_time_index] = completion_time -
processes[min_remaining_time_index].burst_time -
processes[min_remaining_time_index].arrival_time;
            turnaround_time[min_remaining_time_index] = completion_time -
processes[min_remaining_time_index].arrival_time;

            total_waiting_time += waiting_time[min_remaining_time_index];
            total_turnaround_time += turnaround_time[min_remaining_time_index];
        }
    }

    // Print result
    printf("Process\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].burst_time,
               processes[i].arrival_time, waiting_time[i], turnaround_time[i]);
    }

    // Print average waiting time and average turnaround time
    printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
}

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

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

    printf("\nSJF (Preemptive) Scheduling:\n");
    sjf_preemptive(processes, n);

    return 0;
}
```

Output



```
┌──(kali㉿kali)-[~/Desktop/22054029/5th_lab]
└─$ ./a.out
Enter the number of processes: 4
Enter arrival time and burst time for process 1: 1 4
Enter arrival time and burst time for process 2: 2 7
Enter arrival time and burst time for process 3: 5 10
Enter arrival time and burst time for process 4: 6 12

SJF (Preemptive) Scheduling:
Process Burst Time      Arrival Time    Waiting Time    Turnaround Time
1       4               1               0               4
2       7               2               3               10
3       10              5               7               17
4       12              6               16              28
Average Waiting Time: 6.50
Average Turnaround Time: 14.75
```