# EXPERIMENT-4

**Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next**

## AIM:-

To design a CPU scheduling program in C that implements the Shortest Job Next (SJN) or Shortest Job First (SJF) scheduling technique, where the waiting process with the smallest execution time is selected to execute next.

## ALGORITHM:-

1. Input Process Details:
2. Read the number of processes.
3. Input the burst times for each process.
4. Sort Processes by Burst Time:
5. Arrange processes in ascending order of burst time.
6. Calculate Completion Time (CT):
7. For the first process, CT = Burst Time.
8. For subsequent processes, CT = CT(previous) + Burst Time.
9. Calculate Turnaround Time (TAT):
10. TAT = CT - Arrival Time (Assume arrival time is 0).
11. Calculate Waiting Time (WT):
12. WT = TAT - Burst Time.
13. Calculate Average TAT and WT:
14. Compute the average turnaround time and waiting time.
15. Display Results:
16. Show process IDs, burst times, completion times, turnaround times, and waiting times, along with averages.

## CODE:-

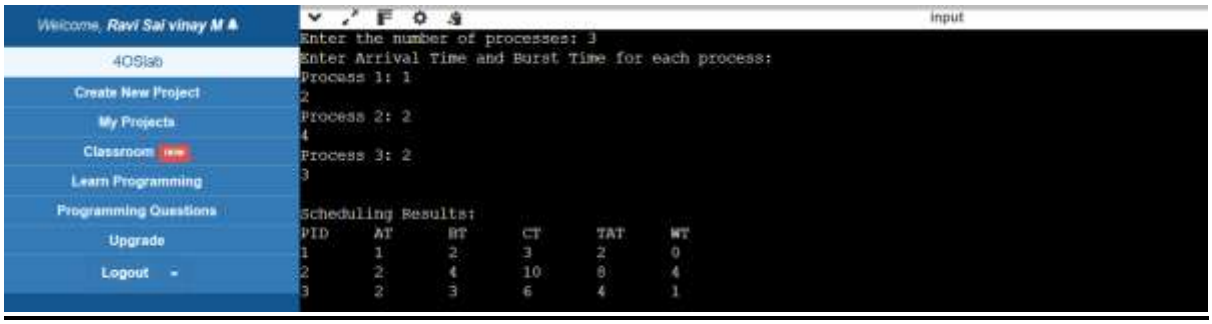
```
#include <stdio.h>


typedef struct {
    int process_id;
    int burst_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
} Process;


void sort_by_burst_time(Process processes[], int n) {
    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) {
                // Swap processes[j] and processes[j + 1]
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}
```

```
int main() {
    int n;
    float avg_turnaround_time = 0, avg_waiting_time = 0;

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

    Process processes[n];

    // Input burst time for each process
    printf("Enter the burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        processes[i].process_id = i + 1;
        printf("Process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
    }

    // Sort processes by burst time
    sort_by_burst_time(processes, n);

    // Calculate Completion Time (CT), Turnaround Time (TAT), and Waiting Time (WT)
    processes[0].completion_time = processes[0].burst_time;
    processes[0].turnaround_time = processes[0].completion_time; // Since Arrival Time = 0
```

```
    processes[0].waiting_time = processes[0].turnaround_time - processes[0].burst_time;

    for (int i = 1; i < n; i++) {
        processes[i].completion_time = processes[i - 1].completion_time + processes[i].burst_time;
        processes[i].turnaround_time = processes[i].completion_time; // Since Arrival Time = 0
        processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
    }

    // Calculate averages
    for (int i = 0; i < n; i++) {
        avg_turnaround_time += processes[i].turnaround_time;
        avg_waiting_time += processes[i].waiting_time;
    }
    avg_turnaround_time /= n;
    avg_waiting_time /= n;

    // Display results
    printf("\nProcess\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting 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].process_id, processes[i].burst_time,
               processes[i].completion_time, processes[i].turnaround_time, processes[i].waiting_time);
    }
```

```
    printf("\nAverage Turnaround Time: %.2f\n", avg_turnaround_time);
    printf("Average Waiting Time: %.2f\n", avg_waiting_time);

    return 0;
}
```

## OUTPUT:-




## RESULT:-

The CPU scheduling program implementing the Shortest Job Next (SJN) technique successfully calculated the completion, turnaround, and waiting times for all processes, along with their averages. The program ensured processes with the smallest burst times were executed first.