Implementation of SJF (Shortest Job First) CPU Scheduling Algorithm

Aim:

To schedule snapshot of processes queued according to SJF scheduling.

Shortest Job First:

- process that requires smallest burst time is processed first.
- SJF can be preemptive or non-preemptive.
- when two processes require same amount of CPU utilization, FCFS is used to break the tie.
- Generally efficient as it results in minimal average waiting time.
- Can result in starvation, since long critical processes may not be processed.

Algorithm:

- Define an array of structure process with members pid, btime, wtime & ttime.
- Get length of the ready queue.
- Obtain btime for each process
- Sort the processes according to their btime in ascending order.

* If two process have same btime, then FCFS is used to resolve the tie.

- The wtime for first process is 0.
- compute wtime and ttime for each proce
  a. $wtime_{i+1} = wtime\,i + btime\,i$
  b. $ttime\,i = wtime\,i + btime\,i$
- compute average waiting time await an average turn around time atur.
- Display btime, ttime and wtime for ea process.
- Display GIANTT chart for the above sch
- Display await and atur.
- Stop

IMPLEMENTATION OF SJF CPU SCHEDULING ALGORITHM

PROGRAM:

```
#include <stdio.h>
struct process
{
        int pid;
        int btime;
        int wtime;
        int ttime;
} p[10], temp;
int main()
{
        int i, j, k, n, ttur, twat;
        float awat,atur;
        printf("Enter no. of process : ");
        scanf("%d", &n);
        for(i=0; i<n; i++)
        {
                printf("Burst time for process P%d (in ms) : ",(i+1));
                scanf("%d", &p[i].btime);
                p[i].pid = i+1;
        }
        for(i=0; i<n-1; i++)
        {
        for(j=i+1; j<n; j++)
```

```
    {
    if((p[i].btime > p[j].btime) || (p[i].btime == p[j].btime  && p[i].pid >      p[j].pid))
    {
            temp = p[i];
            p[i] = p[j];
            p[j] = temp;
    }
    }
    }
    p[0].wtime =0;
    for(i=0; i<n; i++)
    {
            p[i+1].wtime =p[i].wtime + p[i].btime;
            p[i].ttime = p[i].wtime + p[i].btime;
    }
    ttur = twat = 0;
    for(i=0; i<n; i++)
    {
            ttur += p[i].ttime;
            twat += p[i].wtime;
    }
    awat = (float)twat / n;
    atur = (float)ttur / n;
    printf("\n SJF Scheduling\n\n");
    for(i=0; i<28; i++)
            printf("-")
    printf("\nProcess B-Time T-Time W-Time\n");
```

```
        printf("\n0");
        for(i=0; i<n; i++)
        {
                for(j=0; j<p[i].btime; j++)
                printf(" ");
                printf("%2d",p[i].ttime);
        }
}
```

Result

Thus waiting time and turnaround time for processes based on SJF scheduling was computed and the average waiting time was determined.