GOA COLLEGE OF ENGINEERING

FARMAGUDI, PONDA GOA

SHORTEST JOB FIRST

EXPT NO: 2 **DATE:**

AIM

To implement Shortest Job First (SJF) CPU scheduling algorithm.

THEORY

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. These algorithms are either non-pre-emptive or pre-emptive. Non-pre-emptive algorithms are designed so that once a process enters the running state, it cannot be pre-empted until it completes its allotted time, whereas the pre-emptive scheduling is based on priority where a
scheduler may pre-empt a low priority running process anytime when a high priority process enters into a ready state.

Shortest Job First (SJF) Scheduling Algorithm

The shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN, also known as Shortest Job Next (SJN).

- Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.
- It is practically infeasible as Operating System may not know burst times and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times.
- SJF can be used in specialized environments where accurate estimates of running time are available.

Merits:

- SJF is better than the First come first serve (FCFS) algorithm as it reduces the average waiting time.
- SJF is generally used for long term scheduling
- It is suitable for the jobs running in batches, where run times are already known.
- SJF is probably optimal in terms of average turnaround time.

Demerits:

- SJF may cause very long turn-around times or starvation.
- In SJF job completion time must be known earlier, but sometimes it is hard to predict.
- Sometimes, it is complicated to predict the length of the upcoming CPU request.
- It leads to the starvation that does not reduce average turnaround time.

CODE

```
#include<iostream>
#include<algorithm>
#define MAX 10
using namespace std;
int N;

typedef struct PROCESSES{
    int index;
    int BT;
    int CT;
    int TAT;
    int WT;
};

void processInput(PROCESSES Table[])
{
    cout<<"ENTER NUMBER OF ELEMENTS: ";
    cin>>N;
    cout<<"ENTER THE PROCESS DETAILS"<<endl;
    for(int i=0;i<N;i++)
    {
        cout<<"P"<<i+1<<": \n";
        Table[i].index=i+1;
        cout<<"ENTER BURST TIME: ";
        cin>>Table[i].BT;
        cout<<endl;
    }
}

bool CRITERIA1(PROCESSES A,PROCESSES B)
{
    return(A.BT<B.BT);
}

void SJF(PROCESSES Table[])
{
    int sum=0;
    cout<<"\n\nGHANTT CHART: \n";
    sort(Table,Table+N,CRITERIA1);
    for(int i=0;i<N;i++)
    {
        cout<<"|"<<sum<<"|";
        cout<<"---"<<"P"<<Table[i].index<<"---";
        sum+=Table[i].BT;
        Table[i].CT=sum;
    }
    cout<<"|"<<sum<<"|"<<endl;
}

bool CRITERIA2(PROCESSES A,PROCESSES B)
{
    return(A.index<B.index);
}
```

```
void ProcessInfo(PROCESSES Table[])
{
    int TotalTAT=0,TotalWT=0;
    sort(Table,Table+N,CRITERIA2);

    cout<<"\nPROCESS     "<<"BT     "<<"TAT     "<<"WT     "<<endl;
    for(int i=0;i<N;i++)
    {
        cout.setf(ios::left,ios::adjustfield);
        cout.width(12);
        cout<<Table[i].index;

        cout.width(7);
        cout<<Table[i].BT;

        cout.width(8);
        Table[i].TAT=Table[i].CT; //TAT=CT-ArrivalTime
        cout<<Table[i].TAT;
        TotalTAT+=Table[i].TAT;

        cout.width(7);
        Table[i].WT=Table[i].TAT-Table[i].BT;
        cout<<Table[i].WT;
        TotalWT+=Table[i].WT;

        cout<<endl;
    }

    cout<<"\nATAT ="<<TotalTAT/N;
    cout<<"\nAWT ="<<TotalWT/N<<endl;
}

int main()
{
    PROCESSES Table[MAX];
    processInput(Table);
    SJF(Table);
    ProcessInfo(Table);

}
```

OUTPUT

```
ENTER NUMBER OF ELEMENTS: 4
ENTER THE PROCESS DETAILS
P1:
ENTER BURST TIME: 6

P2:
ENTER BURST TIME: 8

P3:
ENTER BURST TIME: 7

P4:
ENTER BURST TIME: 3



GHANTT CHART:
|0|---P4---|3|---P1---|9|---P3---|16|---P2---|24|

PROCESS     BT     TAT     WT
1           6      9       3
2           8      24      16
3           7      16      9
4           3      3       0

ATAT =13
AWT =7
PS C:\Users\Lloyd\Desktop\OS\EXPT2>
```

CONCLUSION

The Shortest Job First CPU Scheduling Algorithm was successfully comprehended and implemented.

REFERENCE

Operating System Concepts
- Abraham Siberschatz, Peter B.Galvin, Gerg Gagne