Experiment no: 02

Experiment Title: Short Job First (Preemptive)

Theory: The shortest job first algorithm prefers processes that arrive first and have the shortest burst times. It can be both a preemptive and non-preemptive algorithm.

Preemptive means that if another process arrives with an even shorter burst time after the process with the shortest burst time and arrival time is given CPU time, the CPU cycle is allocated to the process with the shorter burst time.

Code:

```
#include <stdio.h>
int main ()
{
   int n, c [10], w[10] = {0}, t[10], arrive[10], temp[10], counter = 0, endcounter = 0, timer = 0,
   smalest = 9;
   float avgW = 0, sumW = 0, avgT = 0, sumT = 0;
   printf("Enter the number of processes: ");
   scanf("%d", &n);
   for (int i = 0; i < n; i++)
   {
      printf("\nEnter the CPU Time of P%d: ",i+1);
      scanf("%d", &c[i]);
      temp[i] = c[i];
      printf("Enter the Arrival Time of P%d: ",i+1);
      scanf("%d", &arrive[i]);
   }
   c[smalest] = 9999;
   for (timer = 0; counter != n; timer++)
```

```
    {
        smalest = 9;
        for (int i = 0; i < n; i++)
        {
            if (arrive[i] <= timer && c[i] < c[smalest] && c[i] > 0)
            {
                smalest = i;
            }
        }
        c[smalest]--;
        if (c[smalest] == 0)
        {
            counter++;
            endcounter = timer + 1;
            t[smalest] = endcounter - arrive[smalest];
        }
    }
    printf("\nProcess\tArrival Time\tCPU Time\tWaiting Time\tTurnaround Time");
    printf("\n-------\t ------------\t--------\t------------\t---------------\n");
                for (int i = 0; i < n; i++)
{
    w[i] = t[i] - temp[i];
        printf("P%d      \t%d         \t%d       \t%d          %d\n", i+1, arrive[i], temp[i], w[i],t[i]);
    }
    for (int i = 0; i < n; i++)
{
    sumW += w[i];
        sumT += t[i];
```

```
    }
    avgW = sumW / n;
        avgT = sumT / n;
        printf ("Average Waiting Time: %f", avgW);
        printf ("\nAverage Turnaround Time: %f", avgT);
        return 0;
}
```

Input and Output: -

```
Enter the number of processes: 4

Enter the CPU Time of P1: 7
Enter the Arrival Time of P1: 0

Enter the CPU Time of P2: 4
Enter the Arrival Time of P2: 2

Enter the CPU Time of P3: 1
Enter the Arrival Time of P3: 4

Enter the CPU Time of P4: 4
Enter the Arrival Time of P4: 5

Process Arrival Time    CPU Time        Waiting Time    Turnaround Time
-------  ------------   --------        ------------    ---------------
P1              0               7               9           16
P2              2               4               1           5
P3              4               1               0           1
P4              5               4               2           6
Average Waiting Time: 3.000000
Average Turnaround Time: 7.000000
```

Experiment Title: - Shortest Job First (Non-Primitive).

Theory: - Shortest Job First (SJF) is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be preemptive or non-preemptive.

Non-preemptive Scheduling is used when a process terminates, or a process switch from running to the waiting state. In this scheduling, once the resources (CPU cycles) are allocated to a process, the process holds the CPU till it gets terminated or reaches a waiting state. In the case of non-preemptive scheduling does not interrupt a process running CPU in the middle of the execution. Instead, it waits till the process completes its CPU burst time, and then it can allocate the CPU to another process.

Code:

```
// Short Job First (Non-Preemptive)

#include <stdio.h>

int main ()

{

   int n, i, j, temp1, temp2, temp3;

   int c [10], w [10], t [10], arrive [10], process [10];

   float sum_w, sum_t;

   float avg_w, avg_t;

   printf("Enter the number of processes: ");

   scanf("%d", &n);



   for(i = 0; i < n; i++)

   {

      printf("\nEnter the CPU Time of P%d: ", i+1);

      scanf("%d", &c[i]);


      process[i] = i+1;
```

```
    printf("Enter the Arrival Time of P%d: ", i+1);
    scanf("%d", &arrive[i]);
}
for (i = 1; i < n-1; i++)
{
    for (j = i+1; j < n; j++)
    {
        if(c[i] > c[j])
        {
            temp1 = c[i];
            c[i] = c[j];
            c[j] = temp1;

            temp2 = process[i];
            process[i] = process[j];
            process[j] = temp2;

            temp3 = arrive[i];
            arrive[i] = arrive[j];
            arrive[j] = temp3;
        }
    }
}
w[0] = 0;
for(i = 1; i < n; i++)
```

```
{
   w[i] = w[i-1] + c[i-1];
}

for(i = 1; i < n; i++)
{
   w[i] = w[i] - arrive[i];
}

for(i = 0; i < n; i++)
{
   t[i] = w[i] + c[i];
}
printf("\n\nProcess\t   Arrival Time\tCPU Time\tWaiting Time\tTurnaround Time\n");
for(i = 0; i < n; i++)
{
   printf("P%d\t   %d\t\t%d\t\t%d\t\t%d\n", process[i], arrive[i], c[i], w[i], t[i]);
}
for(i = 0; i < n; i++)
{
   sum_w += w[i];
   sum_t += t[i];
}
avg_w = sum_w/n;
avg_t = sum_t/n;
printf("\nAverage Waiting Time: %.2f", avg_w);
printf("\nAverage Turnaround Time: %.2f", avg_t);
printf("\n\nGrand Chart:\n\n|");
```

```
    for(i = 0; i < n; i++)
    {
        printf(" P%d     |", i+1);
    }
    printf("\n0");
    for(i = 0; i < n; i++)
    {
        printf("       %d", t[i]+arrive[i]);
    }
}
```

Input and Output:

```
Enter the number of processes: 4
CPU/Burst time of P 1 is :10
Arrival time of P  1  is :0

CPU/Burst time of P 2 is :5
Arrival time of P  2  is :1

CPU/Burst time of P 3 is :3
Arrival time of P  3  is :2

CPU/Burst time of P 4 is :6
Arrival time of P  4  is :3

Process    Arrival time        CPU time        Waiting time    Turn Around Time
P3         2                   3               -2              1
P2         1                   5               2               7
P4         3                   6               5               11
P1         0                   10              14              24

Average waiting time: 4.750000
Average Turnaround time: 10.750000

The grand chart :
|    P1    |P2    |P3    |P4    |
 0        3       8        14       24
```