FCFS

Ex. No.: 6(a)
Date: 22/2/25

SHORTEST JOB FIRST

Aim:

To implement the Shortest Job First (SJF) scheduling technique

Algorithm:

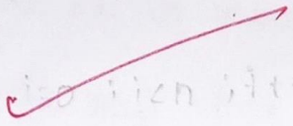
1. Declare the structure and its elements.
2. Get number of processes as input from the user.
3. Read the process name, arrival time and burst time
4. Initialize waiting time, turnaround time & flag of read processes to zero. 5. Sort based on burst time of all processes in ascending order 6. Calculate the waiting time and turnaround time for each process. 7. Calculate the average waiting time and average turnaround time. 8. Display the results.

Program Code:

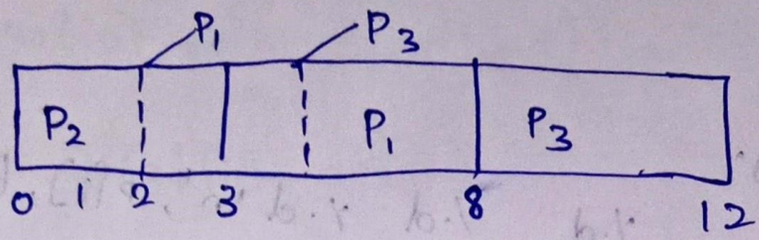
```
#include <stdio.h>
int main() {
 int n;
 Printf("Enter the no of process:\n");
 Scanf("%d",&n);
 int p[n], bt[n];
 Printf("Enter the Burst Time:\n");
 for(int i=0; i<n; i++)
 {
    p[i] = i+1;
    Scanf("%d", &bt[i]);
 }
 for(int i
 Printf("Enter the arrival time:\n");
 for(int i=0; i<n; i++)
 { Scanf("%d", &at[i]);
 }
```

```
for(int i=0 ; i<n-1; i++)
{
   for(int j= i+1 ; j<n; j++)
   {
      if (at[i] > at[j])
      { int t= at[j];
        at[j]= at[i];
        at[i]= t;
        t = p[j];
        p[j] = p[i];
        p[i] = t;
        t = bt[j];
        bt[j]= bt[i];
          bt[i]=t;
      }
   }
}
int wt[n], tat[n]; ct=0;
printf("Completion Time: ");
for(int i=0 ; i<n ; i++)
{
    ct += bt[i];
    printf("%d ", ct);
    tat[i]= ct-at[i];
}
for (int i=0; i<n; i++)
{  wt [i]= tat[i]-bt[i];
}
 float twt=0,ttat=0;
printf("\n");
printf(" Processes          Burst Time   Waiting Time
Turn Around Tim\n");
```

```
for(int i=0 ;i<n ; i++)
{
   twt+ = wt[i];
    ttat+ = tat[i];
   Printf("%d      %d   %d  %d \n", p[i], bt[i],
         wt[i], tat[i]);
 }
 float awt = (float) (twt/n);
 float atat = (float) (ttat/n);
  Printf (" Average Waiting Time = %.1f \n", awt);
  Printf (" Average Turn Around Time = %.1f \n", atat);
}
```



Grant chart



P	BT	AT	CT	TAT (CT-AT)	WT (TAT-BT)
0	5	2	8	6	1
1	3	0	3	3	0
2	4	4	12	8	4

Avg WT = 1.66

Avg TAT = 5.667

Sample Output:

Enter the number of process:
4
Enter the burst time of the processes:
8 4 9 5

Process	Burst Time	Waiting Time	Turn Around Time
2	4	0	4
4	5	4	9
1	8	9	17
3	9	17	26

Average waiting time is: 7.5
Average Turn Around Time is: 13.0

Sample output

Enter the no of process:
3
Enter the Burst time : 5 3 4 , Enter the Arrival time : 2 0 4
Completion Time : 3 8 12

Process	Burst Time	Waiting Time	Turn Around Time
1	3	0	3
0	5	1	6
2	4	4	8

Average Waiting Time = 1.7
Average Turn Around Time = 5.7

Result:

Thus the C program of for First come first Serve is sucessfully executed.

Ex. No.: 6(b)
Date: 22/2/25

Shortest Job First (SJF)

~~**FIRST COME FIRST SERVE**~~

Aim:

To implement First-come First- serve (FCFS) scheduling technique

Algorithm:

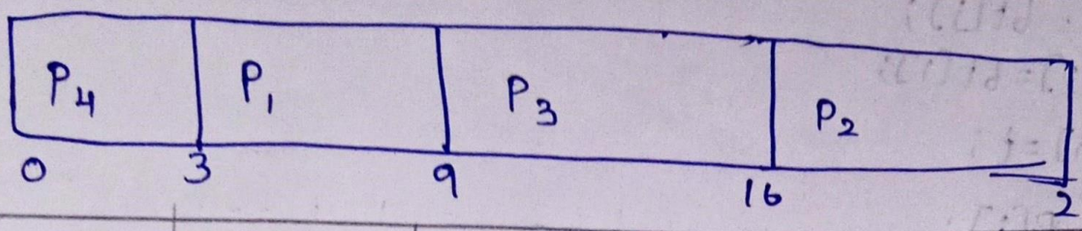
1. Get the number of processes from the user.
2. Read the process name and burst time.
3. Calculate the total process time.
4. Calculate the total waiting time and total turnaround time for each process 5. Display the process name & burst time for each process. 6. Display the total waiting time, average waiting time, turnaround time

Program Code:

```
#include<stdio.h>
int main(){
  int n;
  printf("Enter the number of process:\n");
  Scanf("%d",&n);
  int p[n],bt[n];
  printf("Enter the burst time \n");
  for(int i=0;i<n;i++)
  {
    p[i]=i+1;
    Scanf("%d",&bt[i]);
  }
  for(int i=0;i<n-1;i++)
  { for(int j=i+1;j<n;j++)
    {
      if(bt[i]>bt[j])
```

```
{
    int t = bt[j];
    bt[j] = bt[i];
    bt[i] = t;
    t = P[j];
    P[j] = P[i];
    P[i] = t;
  }
}
int wt[n], tat[n], wt[0] = 0;
for(int i = 1; i < n; i++)
{
    wt[i] = wt[i-1] + bt[i-1];
}
for(int i = 0; i < n; i++)
{
    tat[i] = wt[i] + bt[i];
}
printf("Processes  Burst Time  Waiting Time  Turn Around
    Time\n");
float twt = 0, ttat = 0;
for(int i = 0; i < n; i++)
{
    twt = twt + wt[i];
    ttat = tat[i];
    printf("%d  %d  %d  %d\n", P[i], bt[i], wt[i]
        tat[i]);
}
printf("Average Waiting Time = %.1f", (float)(twt/n));
printf("\n");
printf("Average Turn Around Time = %.1f", (float)
    (ttat/n));
}
```

Gantt Chart



Process	BT (ms)	~~(TAT-BT)~~ CT (ms)	(CT-AT) TAT (ms)	(TAT-BT) WT (ms)
1	6	9	9	3
2	8	24	24	16
3	7	16	16	9
4	3	3	3	0

Avg WT = 7 ms

Avg TAT = 13 ms.

Sample Output:

Enter the number of process:
3
Enter the burst time of the processes:
24 3 3

Process	Burst Time	Waiting Time	Turn Around Time
0	24	0	24
1	3	24	27
2	3	27	30

Average waiting time is: 17.0
Average Turn around Time is: 19.0

Enter the number of Process:
4
Enter the burst time:
6 8 7 3

Process	Burst Time	Waiting Time	Turn Around Time
4	3	0	3
1	6	3	9
3	7	9	16
2	8	16	24

Average waiting time : 7.0
Average Turn Around Time = 13.0

Result:

Thus the C program for Shortest Job first is Successfully executed.

Ex. No.: 6c)
Date: 8/3/25

PRIORITY SCHEDULING

Aim:

To implement priority scheduling technique

Algorithm:

1. Get the number of processes from the user.
2. Read the process name, burst time and priority of process.
3. Sort based on burst time of all processes in ascending order based priority 4.
Calculate the total waiting time and total turnaround time for each process 5.
Display the process name & burst time for each process.
6. Display the total waiting time, average waiting time, turnaround time

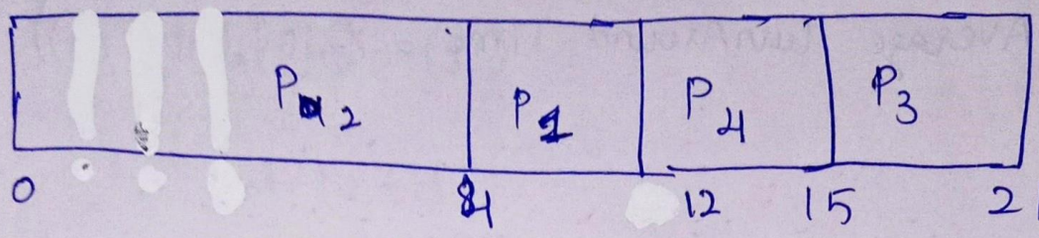
Program Code:

```
#include <stdio.h>
int main(){
  int n;
  printf("Enter the number of process:\n")
  Scanf("%d",&n);
  int p[n];
    int pr[n];
    int bt[n];
    printf("Enter the burst time of the processes\n");
  for(int i=0  i<n ; i++)
  {
     p[i]=i+1;
     Printf("P[%d]\n",i+1);
     printf("Burst Time:");
     Scanf("%d",&bt[i]);
     printf("priority:");
     Scanf("%d",&pr[i]);
  }
```

```
for(int i=0; i<n-1; i++)
 {
   for(int j=i+1 ; j<n ; j++)
   {
       if (pr[i] > pr[j])
        {
           int t= bt[i];
            bt[j] =bt[i];
            bt[i] = t;
            t = p[j];
            p[j] = p[i];
            p[i] = t;
        }
   }
}
int wt[n], tat[n];
wt[0] = 0;
for(int i=1 ; i<n ; i++)
{
   wt[i] = wt[i-1] + bt[i-1];
 }
for(int i=0 ; i<n ; i++)
 {
    tat[i] = wt[i] + bt[i];
 }
printf(" Processes   Burst Time   Waiting Time
        Turn Around Time \n");
float twt = 0, ttat = 0;
for(int i=0 ; i<n ; i++)
 {
    twt = twt + wt[i];
    ttat = ttat + tat[i];
    printf (" %d    %d    %d    %d\n", p[i], bt[i],
            wt[i], tat[i]); }
```

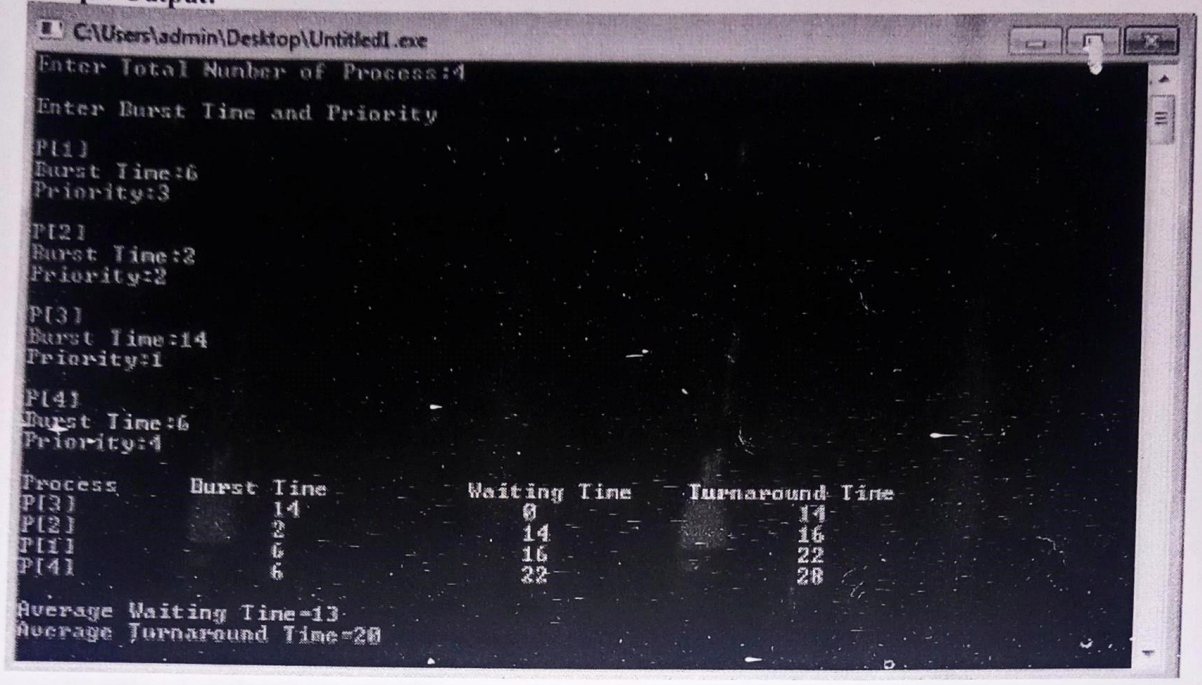
```
printf("Average waiting Time = %.1f", (twt/n));
printf("\n");
printf("Average TurnAround Time = %.1f", (ttat/n));
```

Gantt chart



P	AT (ms)	BT (ms)	CT (ms)	TAT (ms)	WT (ms)
P_1	0	8	12	12	4
P_2	0	4	4	4	0
P_3	0	6	21	21	15
P_4	0	3	15	15	12

Sample Output:



Enter Total Number of process : 4
Enter Burst Time and priority
P[1]
Burst time : 8 priority : 2
P[2]
Burst time : 04 priority : 1
P[3]
Burst time 6 priority : 4
P[4]
Burst time : 3 priority : 3

Process	Burst Time	Waiting Time	Turn Around Time
P_1	8	04	12
P_2	4	0	4
P_3	6	15	21
P_4	3	12	15

Average waiting Time = 7.75
Average turnaround time = 13

Result:
Thus the C program from priority scheduling is successfully executed

Ex. No.: 6d)
Date 21/3/25

ROUND ROBIN SCHEDULING

Aim:

To implement the Round Robin (RR) scheduling technique

Algorithm:

1. Declare the structure and its elements.
2. Get number of processes and Time quantum as input from the user.
3. Read the process name, arrival time and burst time
4. Create an array **rem_bt[]** to keep track of remaining burst time of processes which is initially copy of bt[] (burst times array)
5. Create another array **wt[]** to store waiting times of processes. Initialize this array as 0. 6. Initialize time : t = 0
7. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.
   a- If rem_bt[i] > quantum
   (i) t = t + quantum
   (ii) bt_rem[i] -= quantum;
   b- Else // Last cycle for this process
   (i) t = t + bt_rem[i];
   (ii) wt[i] = t - bt[i]
   (iii) bt_rem[i] = 0; // This process is over
8. Calculate the waiting time and turnaround time for each process.
9. Calculate the average waiting time and average turnaround time.
10. Display the results.

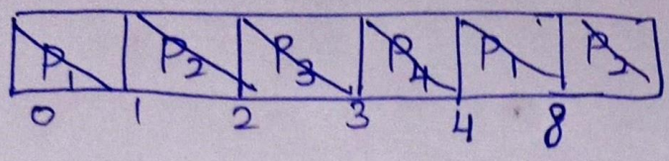
Program Code:

```
#include<stdio.h>
Struct process{
    int pid, at, bt, ct, rem, wt, tat;
};
void sort (struct process* p[], int n)
{
    for(int i=0; i<n-1; i++){
        for(int j=0; j<n; j++){
            if(p[j].at > p[j+1].at){
                struct process t = p[j];
                p[j] = p[j+1];
                p[j+1] = t;
            }
        }
    }
}
```

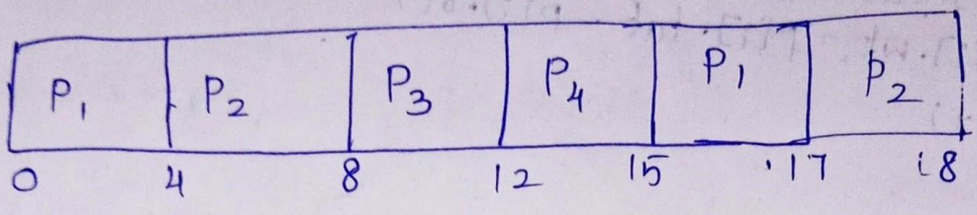
```
int main(){
  int n;
   printf("Enter the number of processes:");
   scanf("%d", &n);
   struct process p[n];
   for(int i=0; i<n;i++){
      printf("Enter Details of Process [%d]\n",i+1);
       p[i].pid = i+1;
       printf("Arrival time:");
       scanf("%d", &p[i].at);
       printf("Burst Time");
       scanf("%d", &p[i].bt);
        p[i].rem = p[i].bt;
        p[i].ct=0;
   }
int q;
printf("Enter the quantum:");
scanf("%d", &q);
int t=0,c=0;
while(c<n)
{
    for(int i=0; i<n; i++)
    {
        if(p[i].rem>q)
        {
          t+=q;
          p[i].rem-=q;
        }
```

```
else if (p[i].ct==0)
{
    t+=p[i].rem;
    p[i].rem=0;
    p[i].ct=t;
    p[i].tat = p[i].ct - p[i].at;
    p[i].wt = p[i].tat - p[i].bt;
    c++;
}
}
}
printf("Process ID    Burst Time    Turn Around Time    waiting Time\n");
float twt=0, ttat=0;
for(int i=0; i<n; i++)
{
    twt=twt+p[i].wt;
    ttat = ttat + p[i].tat;
    Printf("%d    %d    %d    %d\n", p[i].pid, p[i].bt,
        p[i].tat, p[i].wt);
}
Printf("Average Waiting Time = %.1f", (twt/n));
Printf("\n");
Printf("Average Turn Around Time = %.1f", (ttat/n));
}
```

Ready Queue

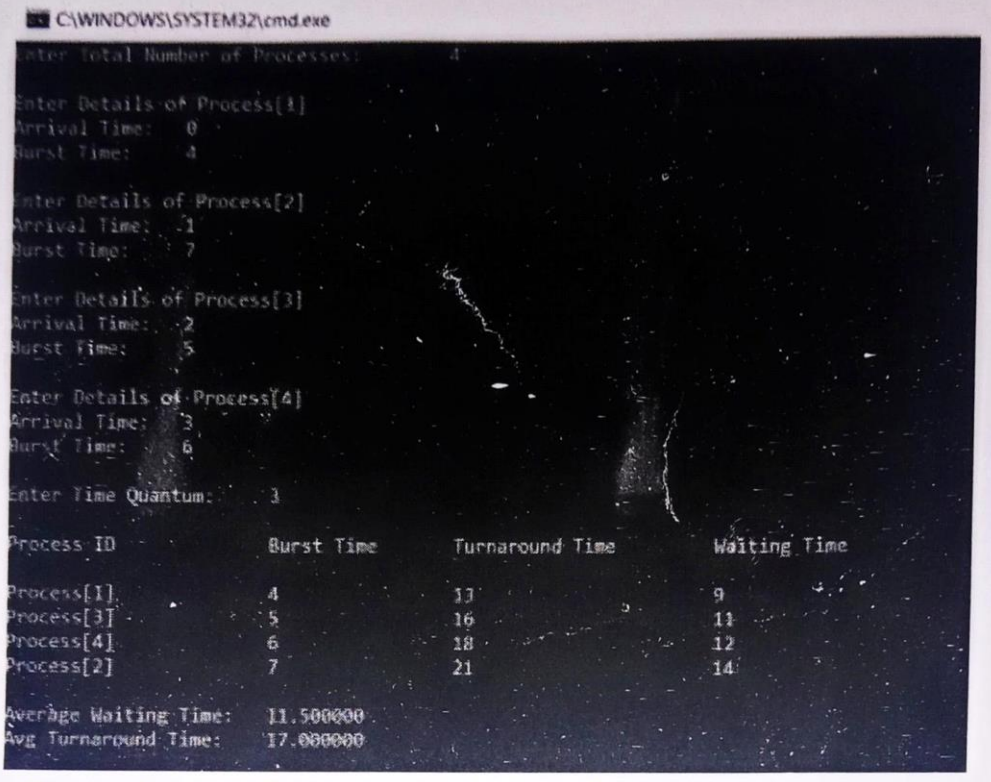


Gantt chart



P	AT (ms)	BT (ms)	CT (ms)	TAT (ms)	WT (ms)
P_1	0	6	17	17	11
P_2	1	5	18	17	12
P_3	2	4	12	10	6
P_4	3	3	15	12	9

Sample Output:



Enter Total number of processes : 4
Enter Details of Process[1]
Arrival Time: 0
Burst Time : 4
Enter Details of Process[2]
Arrival Time : 1
Burst Time : 7
Enter Details of Process[3]
Arrival Time : 2
Burst Time : 4
Enter Details of Process[4]
Arrival Time : 3
Burst Time : 3
Enter Time Quantum : 4

Process ID	Burst Time	Turnaround Time	Waiting Time
P_1	6	17	11
P_2	5	17	12
P_3	4	10	6
P_4	3	12	9

Average waiting Time = 9.5

Average Turnaround Time = 14

Result:

Thus the C program for Round Robin is Successfully executed.