Ex. No.: 6a)
Date: 20/2/25

FIRST COME FIRST SERVE

Aim:

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

Algorithm:

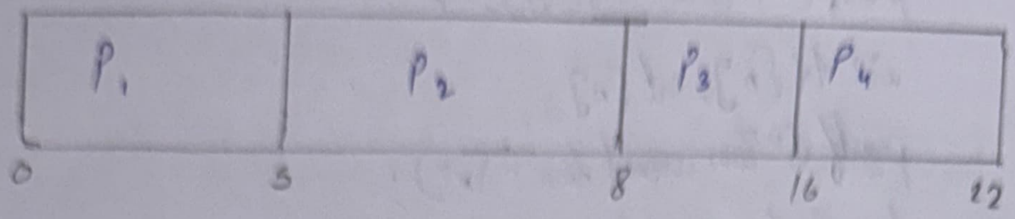
1. Get the number of processes from the user.
2. Read the process name and burst time.
3. Calculate the total process time.
4. Calculate the total waiting time and total turnaround time for each process 5. Display the process name & burst time for each process. 6. Display the total waiting time, average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

Program Code:

```
#include<stdio.h>
int main(){
  int num;
  printf("Enter the number of process:");
  Scanf("%d",&num);
  int bt[n];
  printf("Burst time:");
  for(int i=0; i<n; i++){
     Scanf("%d",&bt[i]);}
  int ct[n];
  printf("Completion time:");
  int Cont=0;
  for(int i=0; i<n; i++){
```

```
Count+ = bt[i];
Count t[i]= Count;
printf("%d\n", ct[i]); }
int tt[n], int[n];
printf("Turn around time : \n");
for (int i=0; i<n; i++) {
 tt[i] = ct[i];
 printf("%d\n", tt[i]); }
printf("Waiting time : \n");
 for (int i=0; i<n; i++) {
     wt[i] = tt[i] - bt[i];
   printf(".%d\n", wt[i]; }
int avg-wt = 0, avg-tt = 0;
for (int i=0; i<n; i++) {
    avg-wt += wt[i];
    avg-tt += tt[i]; }
   avg-wt = avg-wt/n;
  avg-tt = avg-tt/n;
 printf("Average waiting Time : %d\n", avg-wt);
printf("Average turn around Time : %d\n", avg-tt);
}.
```

Gantt chart



Process	Burst Time	Waiting Time	Turn Around Time
P_1	5	0	5
P_2	3	5	8
P_3	8	8	16
P_4	6	16	22

Average Waiting Time = 7.25 ms

Average Turnaround Time = 12.75 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
```

Average Waiting Time : 7.25

Average Turnaround Time : 12.25

Output :-

```
Enter the number of processes : 4

Enter the process names : P1  P2  P3  P4

Enter the burst time of the processes :  5 3 8 6

Process     Burst Time      Waiting Time      Turn Around Time
P1              5                0                  5
P2              3                5                  8
P3              8                8                 16
P4              6               16                 22
```

Result:

Hence the fcfs (first come first serve) Scheduling is Verified.

Ex. No.: 6b)
Date: 26/2/25

SHORTEST JOB FIRST

Aim:

To implement the Shortest Job First (SJF) scheduling technique

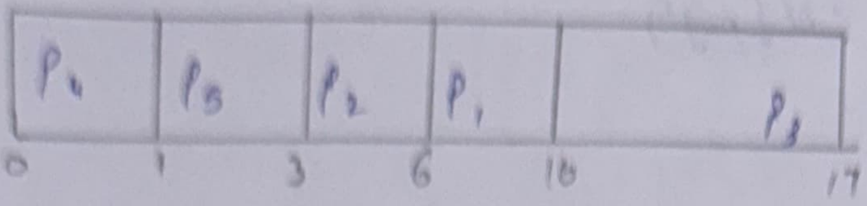
Algorithm:

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 bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
float avg_wt,avg_tat;
printf("Enter number of process:\n");
Scanf("%d",&n);
Printf("Enter Burst Time:n");
for(i=0;i<n;i++)
{
  printf("%d",&bt[i]);
  Scanf("%d",&bt[i]);
  P[i]=i+1;
  }
  for(i=0;i<n;i++)
  {
   pos=i;
   for(j=i+1;j<n;j++)
{
if (bt[j] < bt[pos])
    Pos=j;
}
 tmp = bt[i];
 bt[i] = bt[pos];
 bt[pos] = tmp;
  tmp = p[i];
  p[i] = p[pos]
  p[pos] = tmp;
 }
 wt[0] = 0;
for (i=1; i<n; i++)
{
  wt[i] = 0
  for [j=0 ; j<1 ; j++)
     wt[i] += bt[j];
  total += wt[i];
 }
 avg_wt = (float) total / n;
 total = 0;
printf ("Process  Burst Time   waiting Time   Turn around Time)
for (i=0; i<n; i++)
{
   tat[i] = bt[i] + wt[i];
   total += tat[i]
   printf ("np-%d\t %d\t  %d\t\t %d", p[i] bt[i], wt[i],
                                                          tat[i]);
   }
   avg . tat = (float) total / n;
   printf ("n Average waiting time = %f", avg_wt);
  printf ("n Average Turn around Time = %f n", avg_tat);
}
```

Gantt chart



Process	Burst Time	Waiting Time	Turnaround Time
P4	1	0	1
P5	2	1	3
P2	3	3	6
P1	4	6	10
P3	7	10	17

Average Waiting time = 4.00ms

Average Turn around time = 7.4ms

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
```

```
Average Waiting time = 4.000000
Average Turnaround Time = 7.400000
```

```
Enter number of process : 5
Enter Burst Time :   4 3 7 1 2
Process      Burst Time      Waiting Time      Turnaround Time
 P4             1                0                  1
 P5             2                1                  3
 P2             3                3                  6
 P1             4                6                 10
 P3             7               10                 17
```

Result:

Thus the Code to implement the Short job first (SJF) Scheduling technique is executed Successfully.

Ex. No.: 6c)
Date: 06/03/25

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:

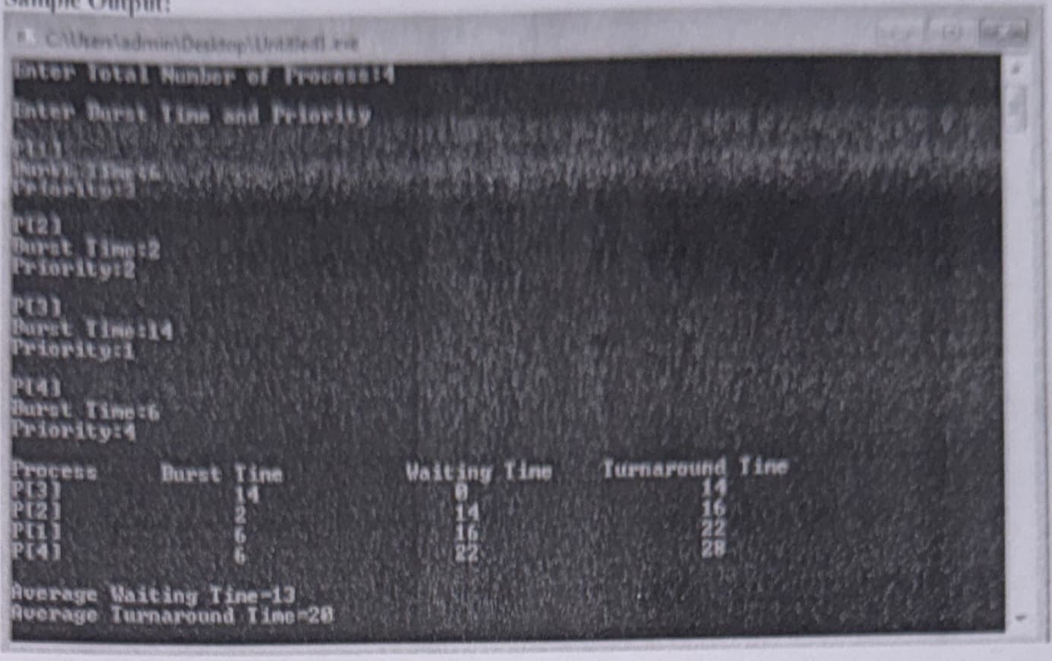
```
# include <stdio.h>
 void swap()
 {
  int temp = *a;
  *a = *b;
  *b = temp;
 }
 int main()
 {
  int n;
  printf("Enter the number of processes: ");
  scanf("%d", &n);
  int b[n], p[n], index[n], index2[n];
  printf("Enter Burst Time: ")
  for(int i=0; i<n; i++)
  {
   scanf("%d", &b[i]);
   index[i] = i+1;
  }
printf ("Enter Priority Values : ");
  for (int i=0; i<n; i++){
     Scanf ("%d", & p[i]);
     index [i] = i+1;
  }
  for (int i=0; i<n; i++)
  {
   int a=p[i];
   int Val=i;
  for (int j=i; j<n ;j++)
  {
   if (p[j] > a)
   {
      a= p[j];
      Val=j;
   }
 }
  Swap (& p[i], & p [m]);
 Swap (& b[i], & b [m]);
 Swap (& index [i], & index [m]);
}
printf ("Process ID   Burst Time   Waiting time   Turn Around Time \n");
int wait_time =0, avg_wt =0, avg_tat =0;
for (int i=0; i<n; i++)
{
 printf ("P %d  %d  %d  %d\n", index[i], b[i], wait_time, wait_time+b[i]);
 wait_time += b[i];
 avg_wt += wait_time;
 avg_tat += (wait_time+=b[i]);
}
 printf ("Average TurnAround Time : %d", avg_tat /4).
 printf ("Average Waiting Time : %d", avg_wt /4);
}
```

Process	Burst time	Waiting time	Turn Around Time
P[2]	8	0	8
P[1]	5	8	13
P[4]	6	13	19
P[3]	3	19	22

Average Waiting time = 10

Average Turn Around time = 15.

Sample Output:



C:\Users\admin\Desktop\Untitled1.exe

```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:6
Priority:3

P[2]
Burst Time:2
Priority:2

P[3]
Burst Time:14
Priority:1

P[4]
Burst Time:6
Priority:4

Process     Burst Time          Waiting Time    Turnaround Time
P[3]              14                0                  14
P[2]              2                 14                 16
P[1]              6                 16                 22
P[4]              6                 22                 28

Average Waiting Time=13
Average Turnaround Time=20
```

Enter the numbers of process : 4

Enter the Burst Time : 5 8 3 6

Priority = 2 1 4 3

Result:

Thus the priority Scheduling technique is implemented

Ex. No.: 6d)
Date 2d3/25

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
```

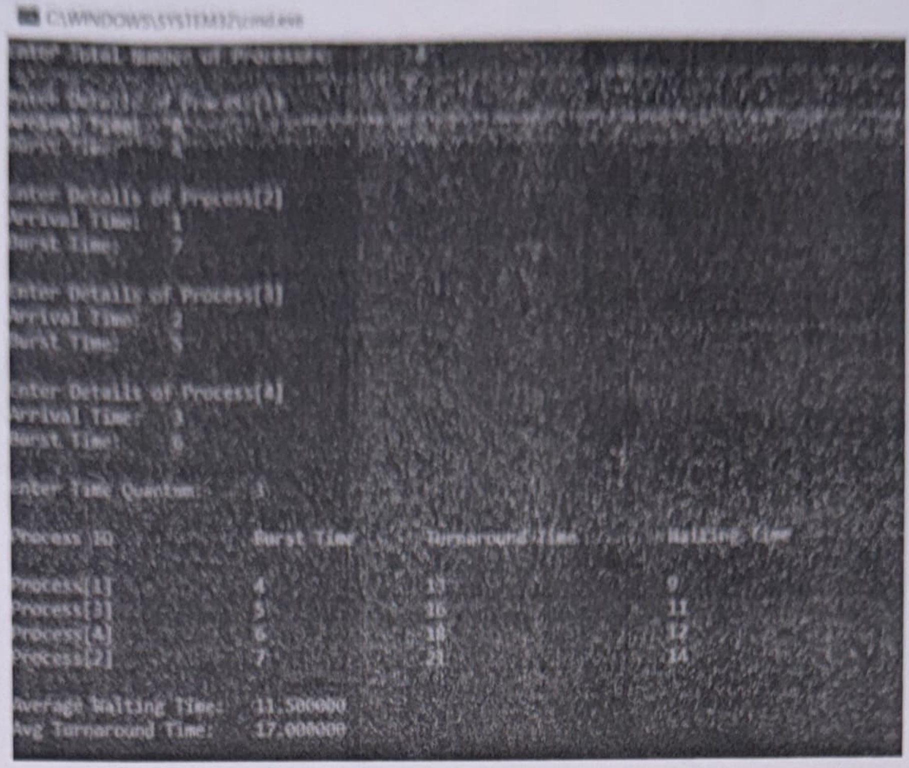
8. Calculate the waiting time and turnaround time for each process.
9. Calculate the average waiting time and average turnaround time.
10. Display the results.

Program Code:

```
# include <stdio.h >
int main () {
    int n, quantum;
    printf(" Enter number of processes: ");
    Scanf ("%d", & n);
    int processes [n], bt [n], at [n], wt [n], tat [n], rem_bt [n];
    printf (" Enter the time Quantum: ");
    Scanf ("%d", & quantum);
    for (int i = 0; i < 2; i++ ) {
```

```
if (i==0) { printf ("Enter Arrival Time ")
else { printf (" Enter Burst Time ")
for (int j=0 ; j<n; j++ ) {
    process [j] = i+1 ;
    if (i==0) { Scanf ("%d", &at [j] ); }
    else { Scanf ("%d", &bt [j] );
        rem_bt [i] = bt [i]; }
    wt [i] = 0
  }
  int t=0
  int Cont ;
  do {
    Cont = 1;
    for (int i=0 ; i<n ; i++) {
      if (rem_bt [i] >0) {
        Cont =0;
        if (rem_bt [i] > quantum ) {
            t+= quantum ;
            rem_bt [i] = quantum ;
        }
        else {
            t+ = rem_bt [i];
            wt [i] = t - bt [i] - at [i];
            rem_bt [i] = 0
        }
      }
    }
  } while (! Cont)
float total_wt = 0, total_tat = 0;
printf("\n process \t AT \t BT \t WT \t TAT\n");
for (int i = 0; i<n; i++){
    tat[i] = bt[i]  + wt[i];
    total_wt += wt[i]
    total_tat  += tat [i];
    printf(" P%d \t %d \t %d \t %d \t %d\n", process[i],
              at[i], bt[i] , wt[i], tat [i]);
  }
printf("\n Average Waiting Time : %.2f", total_wt /n );
printf("\n Average Turnaround Time : %.2f\n", total_tat /n);
return 0;
}
```

Sample Output:



```
nter Total Number of Processes:     4

nter Details of Process[1]
rrival Time:   0
urst Time:     4

nter Details of Process[2]
rrival Time:   1
urst Time:     7

nter Details of Process[3]
rrival Time:   2
urst Time:     5

nter Details of Process[4]
rrival Time:   3
urst Time:     6

nter Time Quantum:     3

rocess ID              Burst Time        Turnaround Time        Waiting Time

rocess[1]              4                 13                     9
rocess[3]              5                 16                     11
rocess[4]              6                 18                     12
rocess[2]              7                 21                     14

verage Waiting Time:   11.500000
vg Turnaround Time:    17.000000
```

Input

Enter the number of process :

Enter Time Quantum : 2

Enter Arrival Time : 3 0 2 9

Enter Burst Time : 5 7 1 9

Output:

```
Process    AT    BT    WT    TAT
  P1        3     5     6     11
  P2        0     7    12     19
  P3        2     1     2      3
  P4        9     9     4     13

Average Waiting Time : 6.00
Average Turn Around Time : 11.50
```

Result:

Thus, Round Robin Scheduling technique is implemented.