Ex. No. 6 (a)	**FIRST COME FIRST SERVE**
Date: 20.02.2025	

Aim:

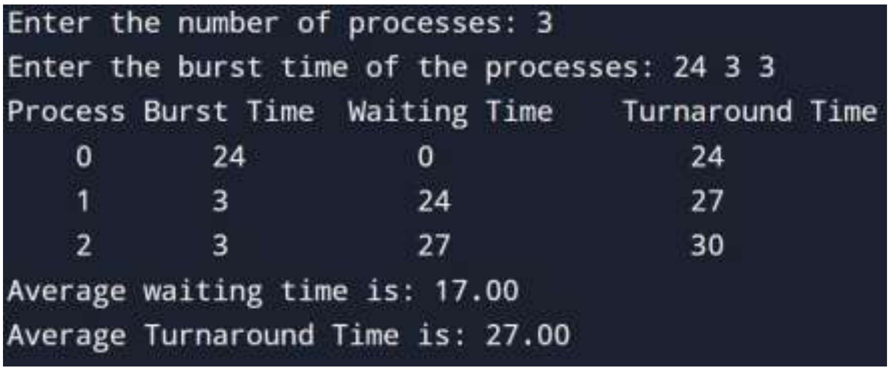
To implement First-come First- serve (FCFS) scheduling.

Program:

```
#include <stdio.h>

void Calculate(int p, int BT[], int WT[], int TAT[]) {
   WT[0] = 0;
   TAT[0] = BT[0] + WT[0];

   for (int i = 1; i < p; i++) {
      WT[i] = WT[i - 1] + BT[i - 1];
      TAT[i] = BT[i] + WT[i];
   }
}

void Display(int p, int BT[], int WT[], int TAT[]) {     printf("\nProcess
BurstTime  WaitingTime  TurnAroundTime\n");

   float totalWT = 0, totalTAT = 0;
   for (int i = 0; i < p; i++) {        printf("%4d %10d %11d %13d\n",
i + 1, BT[i], WT[i], TAT[i]);

      totalWT += WT[i];
      totalTAT += TAT[i];
   }

   printf("\nAverage Waiting Time is: %.1f\n", totalWT / p);
printf("Average Turn Around Time is: %.1f\n", totalTAT / p);
}

void main() {
   int p;
   printf("Enter the number of processes: ");
scanf("%d", &p);
```

```
    int BT[p], WT[p], TAT[p];     printf("Enter the
burst time of the processes: ");
    for (int i = 0; i < p; i++) {
        scanf("%d", &BT[i]);
    }

    Calculate(p, BT, WT, TAT);
Display(p, BT, WT, TAT);
}
```

Output:



```
Enter the number of processes: 3
Enter the burst time of the processes: 24 3 3
Process Burst Time  Waiting Time    Turnaround Time
    0       24          0               24
    1       3           24              27
    2       3           27              30
Average waiting time is: 17.00
Average Turnaround Time is: 27.00
```

Result:

The program implements the First-Come-First-Serve (FCFS) scheduling technique, calculating the waiting time, turnaround time, and averages and executed successfully.

Ex. No. 6 (b)	**SHORTEST JOB FIRST**
Date: 20.02.2025	

Aim:

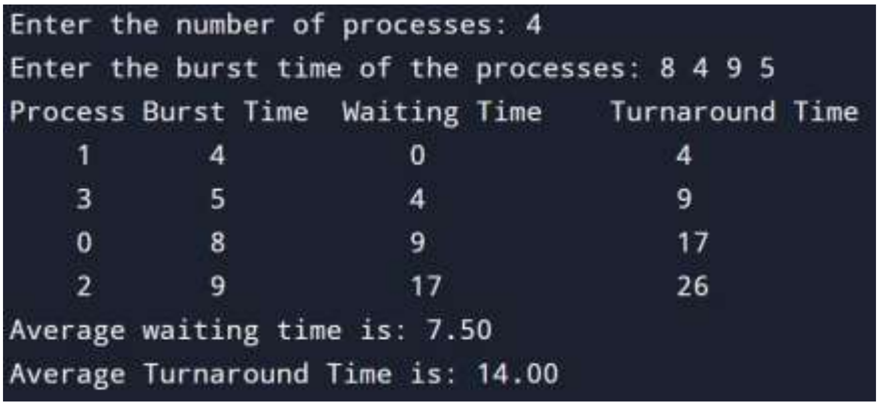
To implement the Shortest Job First (SJF) scheduling.

Program:

```
#include <stdio.h>

void swap(int *a, int *b) {
int temp = *a;     *a = *b;
   *b = temp;
}

void SJFSort(int p, int PC[], int BT[]) {
for (int i = 0; i < p - 1; i++) {        int
minIndex = i;        for (int j = i + 1; j <
p; j++) {            if (BT[j] <
BT[minIndex]) {
            minIndex = j;
         }
      }
      swap(&BT[i],&BT[minIndex]);
      swap(&PC[i],&PC[minIndex]);
   }
}

void Calculate(int p, int BT[], int WT[], int TAT[]) {
   WT[0] = 0;
   TAT[0] = BT[0] + WT[0];

   for (int i = 1; i < p; i++) {
      WT[i] = WT[i - 1] + BT[i - 1];
      TAT[i] = BT[i] + WT[i];
   }
}

void Display(int p, int PC[], int BT[], int WT[], int TAT[]) {
printf("\nProcess  BurstTime  WaitingTime  TurnAroundTime\n");

   float totalWT = 0, totalTAT = 0;
   for (int i = 0; i < p; i++) {        printf("%4d %10d %11d %13d\n",
PC[i], BT[i], WT[i], TAT[i]);

      totalWT += WT[i];
      totalTAT += TAT[i];
   }

   printf("\nAverage Waiting Time is: %.2f\n", totalWT / p);
printf("Average Turn Around Time is: %.2f\n", totalTAT / p);
}
```

```
void main() {
   int p;
   printf("Enter the number of processes: ");
scanf("%d", &p);

   int PC[p], BT[p], WT[p], TAT[p];     printf("Enter
the burst time of the processes: ");
   for (int i = 0; i < p; i++) {
PC[i] = i+1;
      scanf("%d", &BT[i]);
   }
   SJFSort(p, PC, BT);
   Calculate(p, BT, WT, TAT);
   Display(p, PC, BT, WT, TAT);
}
```

Output:



```
Enter the number of processes: 4
Enter the burst time of the processes: 8 4 9 5
Process Burst Time  Waiting Time    Turnaround Time
    1       4           0               4
    3       5           4               9
    0       8           9               17
    2       9           17              26
Average waiting time is: 7.50
Average Turnaround Time is: 14.00
```

Result:

The program implements the Shortest Job First (SJF) scheduling technique, calculating the waiting time, turnaround time, and averages, and executed successfully.

Ex. No. 6 (c)	**PRIORITY SCHEDULING**
Date: 21.02.2025	

Aim:

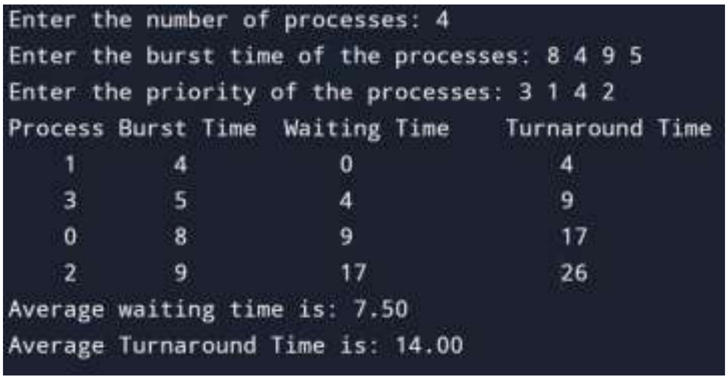
To implement priority scheduling technique.

Program:

```
#include <stdio.h>

void swap(int *a, int *b) {
int temp = *a;     *a = *b;
   *b = temp;
}

void ATSort(int p, int PC[], int BT[], int PR[]) {
   for (int i = 0; i < p - 1; i++) {
int minIndex = i;        for (int j = i
+ 1; j < p; j++) {            if (PR[j] <
PR[minIndex]) {
            minIndex = j;
         }
      }
      swap(&PR[i],&PR[minIndex]);
swap(&BT[i],&BT[minIndex]);
      swap(&PC[i],&PC[minIndex]);
   }
}

void Calculate(int p, int BT[], int WT[], int TAT[]) {
   WT[0] = 0;
   TAT[0] = BT[0] + WT[0];

   for (int i = 1; i < p; i++) {
      WT[i] = WT[i - 1] + BT[i - 1];
      TAT[i] = BT[i] + WT[i];
   }
}

void Display(int p, int PC[], int BT[], int WT[], int TAT[]) {
printf("\nProcess  BurstTime  WaitingTime  TurnAroundTime\n");

   float totalWT = 0, totalTAT = 0;
   for (int i = 0; i < p; i++) {
      printf("%4d %10d %11d %13d\n", PC[i], BT[i], WT[i], TAT[i]);

      totalWT += WT[i];
      totalTAT += TAT[i];
   }

   printf("\nAverage Waiting Time is: %.1f\n", totalWT / p);
printf("Average Turn Around Time is: %.1f\n", totalTAT / p);
}

void main() {
   int p;
   printf("Enter the number of processes: ");
scanf("%d", &p);
```

```
    int PC[p], PR[p], BT[p], WT[p], TAT[p];     printf("\nEnter the
burst time and priority of the processes:\n");     for (int i = 0; i < p;
i++) {         PC[i] = i+1;
        printf("\nP[%d]",PC[i]);
printf("\nBurst     Time:     ");
scanf("%d",               &BT[i]);
printf("Priority: ");
        scanf("%d", &PR[i]);
    }
    ATSort(p, PC, BT, PR);
    Calculate(p, BT, WT, TAT);
    Display(p, PC, BT, WT, TAT);
}
```

Output:



```
Enter the number of processes: 4
Enter the burst time of the processes: 8 4 9 5
Enter the priority of the processes: 3 1 4 2
Process Burst Time  Waiting Time    Turnaround Time
    1       4           0               4
    3       5           4               9
    0       8           9               17
    2       9           17              26
Average waiting time is: 7.50
Average Turnaround Time is: 14.00
```

Result:

The program implements the Priority Scheduling technique, calculating waiting time, turnaround time, and averages, and executed successfully.

Ex. No. 6 (d)	**ROUND ROBIN SCHEDULING**
Date: 21.02.2025	

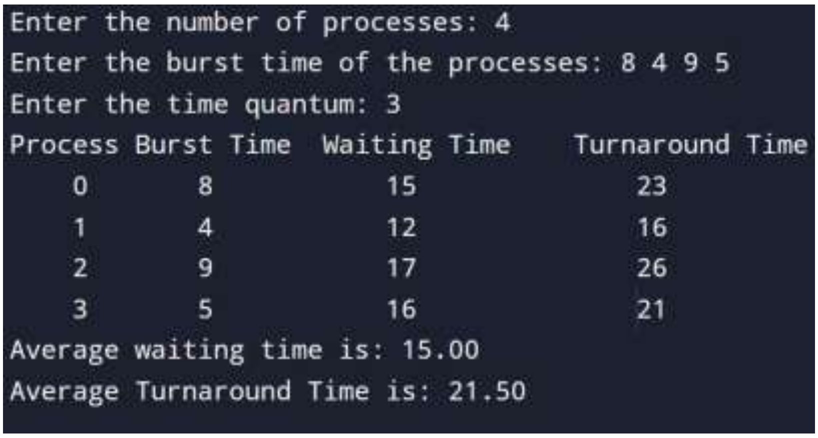
Aim:

To implement the Round Robin (RR) scheduling technique.

Program:

```
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
*a = *b;
    *b = temp;
}

int isEmpty(int p, int RT[]) {
for (int i = 0; i < p; i++) {
        if (RT[i] != 0) return 0;
    }
    return 1;
}

void CTSort(int p, int PC[], int AT[], int BT[], int CT[], int WT[], int TAT[]) {
for (int i = 0; i < p - 1; i++) {        int minIndex = i;        for (int j = i + 1; j < p;
j++) {            if (CT[j] < CT[minIndex]) {
                minIndex = j;
            }
        }
        swap(&PC[i], &PC[minIndex]);
swap(&AT[i], &AT[minIndex]);        swap(&BT[i],
&BT[minIndex]);        swap(&CT[i],
&CT[minIndex]);        swap(&WT[i],
&WT[minIndex]);
        swap(&TAT[i], &TAT[minIndex]);
    }
}

void RRSCalculate(int p, int q, int AT[], int BT[], int CT[], int WT[], int TAT[]) {
int t = 0, completed = 0;    int RT[p];

    for (int i = 0; i < p; i++) RT[i] = BT[i];

    while (completed < p) {
        int idle = 1;

        for (int i = 0; i < p; i++) {
            if (RT[i] > 0 && AT[i] <= t) {
idle = 0;

                if (RT[i] > q) {
t += q;
                    RT[i] -= q;
} else {                t +=
RT[i];                CT[i]
= t;                RT[i] =
0;
                    completed++;
                }
            }
        }
    if (idle) t++;
  }

  for (int i = 0; i < p; i++) {
TAT[i] = CT[i] - AT[i];
    WT[i] = TAT[i] - BT[i];
  }
}

void Display(int p, int PC[], int AT[], int BT[], int CT[], int WT[], int TAT[]) {
printf("\nProcess  ArrivalTime  BurstTime  CompletedTime  TurnAroundTime
WaitingTime\n");

  float totalWT = 0, totalTAT = 0;
for (int i = 0; i < p; i++) {
    printf("%4d %10d %11d %13d %14d %14d\n", PC[i], AT[i], BT[i], CT[i], TAT[i],
WT[i]);
    totalWT += WT[i];
    totalTAT += TAT[i];
  }

  printf("\nAverage Waiting Time is: %.2f\n", totalWT / p);
printf("Average Turn Around Time is: %.2f\n", totalTAT / p);
}

void main() {
  int p, q;
  printf("Enter total number of processes: ");
scanf("%d", &p);

  int PC[p], AT[p], BT[p], CT[p], WT[p], TAT[p];

  printf("\nEnter the arrival time and burst time of the processes:\n");
for (int i = 0; i < p; i++) {       PC[i] = i + 1;
    printf("\nP[%d]\n",  PC[i]);
printf("Arrival       Time:       ");
scanf("%d",                 &AT[i]);
printf("Burst Time: ");
    scanf("%d", &BT[i]);
  }

  printf("\nEnter Time Quantum: ");
  scanf("%d", &q);

  RRSCalculate(p, q, AT, BT, CT, WT, TAT);
  CTSort(p, PC, AT, BT, CT, WT, TAT);
  Display(p, PC, AT, BT, CT, WT, TAT);
}
```

Output:



```
Enter the number of processes: 4
Enter the burst time of the processes: 8 4 9 5
Enter the time quantum: 3
Process Burst Time  Waiting Time    Turnaround Time
    0       8           15              23
    1       4           12              16
    2       9           17              26
    3       5           16              21
Average waiting time is: 15.00
Average Turnaround Time is: 21.50
```

Result:

The program implements the Round Robin Scheduling technique, calculates waiting time, turnaround time, averages, and executed successfully.