CPU SCHEDULING ALGORITHMS

5. Round Robin

```
// CPU Scheduling -Round Robin
#include<stdio.h>
struct process
{
        char name;
        int at,bt,wt,tt,rt;
        int completed;
}p[10];

int n;
int q[10];  //queue
int front=-1,rear=-1;
void enqueue(int i)
{
    if(rear==10)
        printf("overflow");
    rear++;
    q[rear]=i;
    if(front==-1)
        front=0;

}

int dequeue()
{
    if(front==-1)
        printf("underflow");
    int temp=q[front];
    if(front==rear)
        front=rear=-1;
    else
        front++;
    return temp;
}
int isInQueue(int i)
{
    int k;
    for(k=front;k<=rear;k++)
    {
        if(q[k]==i)
        return 1;
    }
    return 0;
}

void sortByArrival()
{
    struct process temp;
    int i,j;
    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(p[i].at>p[j].at)
            {
                temp=p[i];
                p[i]=p[j];
                p[j]=temp;
            }
        }
    }
}

void main()
{
    int i,j,time=0,sum_bt=0,tq;
    char c;
    float avgwt=0;
     printf("\nEnter Number of Processes:\n");
     scanf("%d",&n);
     for(i=0,c='A';i<n;i++,c++)
     {
         p[i].name=c;
         printf("\n Process %c\n",c);
         printf("\n Arrival Time :");
         scanf("%d",&p[i].at);
         printf(" Burst Time :");
         scanf("%d",&p[i].bt);
         p[i].rt=p[i].bt;
         p[i].completed=0;
         sum_bt+=p[i].bt;
    }
    printf("\nEnter the time quantum:");
    scanf("%d",&tq);
    sortByArrival();
    enqueue(0);          // enqueue the first
process
    printf("Process execution order: ");
    for(time=p[0].at;time<sum_bt;)       // run
until the total burst time reached
    {
        i=dequeue();
        if(p[i].rt<=tq)
        {                          /* for processes having
remaining time with less than
or  equal  to time quantum  */

            time+=p[i].rt;
            p[i].rt=0;
            p[i].completed=1;
            printf(" %c ",p[i].name);
            p[i].wt=time-p[i].at-p[i].bt;
            p[i].tt=time-p[i].at;
            for(j=0;j<n;j++)     /*enqueue the
processes which have come while
scheduling */
            {
                if(p[j].at<=time && p[j].completed!
```

```
=1&& isInQueue(j)!=1)
                {
                    enqueue(j);
                }
            }
        }
        else                // more than time
quantum
        {
            time+=tq;
            p[i].rt-=tq;
            printf(" %c ",p[i].name);
            for(j=0;j<n;j++)    /*first enqueue the
processes which have come while
scheduling */
            {
                if(p[j].at<=time && p[j].completed!
=1&&i!=j&& isInQueue(j)!=1)
                {
                    enqueue(j);
                }
            }
            enqueue(i);   // then enqueue the
uncompleted process
        }
    }

    printf("\nName\tArrival Time\tBurst
Time\tResponse Time\tTurnAround Time");
    for(i=0;i<n;i++)
    {
        avgwt+=p[i].wt;
        printf("\n%c\t\t%d\t\t%d\t\t%d\t\t
%d",p[i].name,p[i].at,p[i].bt,p[i].wt,p[i].tt);
    }
    printf("\nAverage waiting time:
%f\n",avgwt/n);
}
```