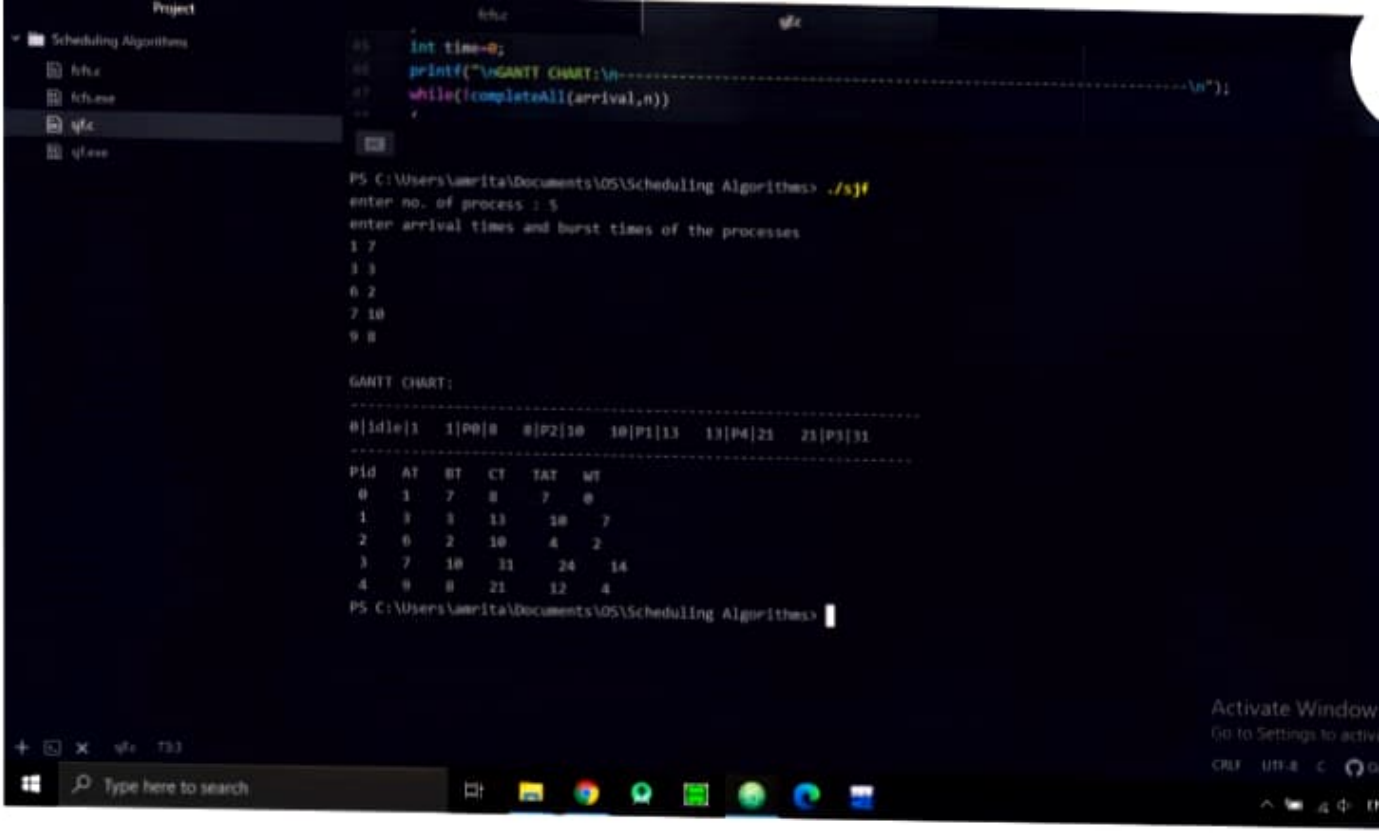
SOURCE CODE:

```
#include<stdio.h>
#include <stdlib.h>
int min(int a[], int bt[], int time, int n)
{
int i,j=-1;
int minimum=32625;
```

```
for(i=0;i<n;i++)
{
if(a[i]!=-1 && a[i]<=time)
{
if(bt[i]<minimum)
{
minimum=bt[i];
j=i;
}
}
}
return j;
}
int completeAll(int a[],int n)
{
int i;
for(i=0;i<n;i++)
{
if(a[i]!=-1)
return 0;
}
return 1;
}
void main()
{
int i,j,k,n;
printf("enter no. of process : ");
scanf("%d",&n);
int arrival[n],ct[n],bt[n],wt[n],turnAr[n],arp[n];
printf("enter arrival times and burst times of the processes \n");
for(i=0;i<n;i++)
{
scanf("%d%d",&arrival[i],&bt[i]);
arp[i]=arrival[i];
}
int time=0;
printf("\nGANTT
CHART:\n----------------------------------------------------------------\n");
while(!completeAll(arrival,n))
{
k=min(arrival,bt,time,n);
```

```
if(k==-1)
{
printf("%d|idle|%d ",time,time+1 );
time++;
continue;
}
else
{
printf("%d",time);
ct[k]=time+bt[k];
turnAr[k]=ct[k]-arrival[k];
wt[k]=turnAr[k]-bt[k];
time+=bt[k];
printf("|P%d|%d ",k,time );
arrival[k]=-1;
}
}
printf("\n----------------------------------------------------------------\n");
printf("Pid AT BT CT TAT WT\n");
for(i=0;i<n;i++)
{
printf(" %d %d %d %d %d
%d\n",i,arp[i],bt[i],ct[i],turnAr[i],wt[i] );
}
}
```

OUTPUT:



3. Implement Non-Pre-emptive Priority Based job sched algorithm in C programming language.

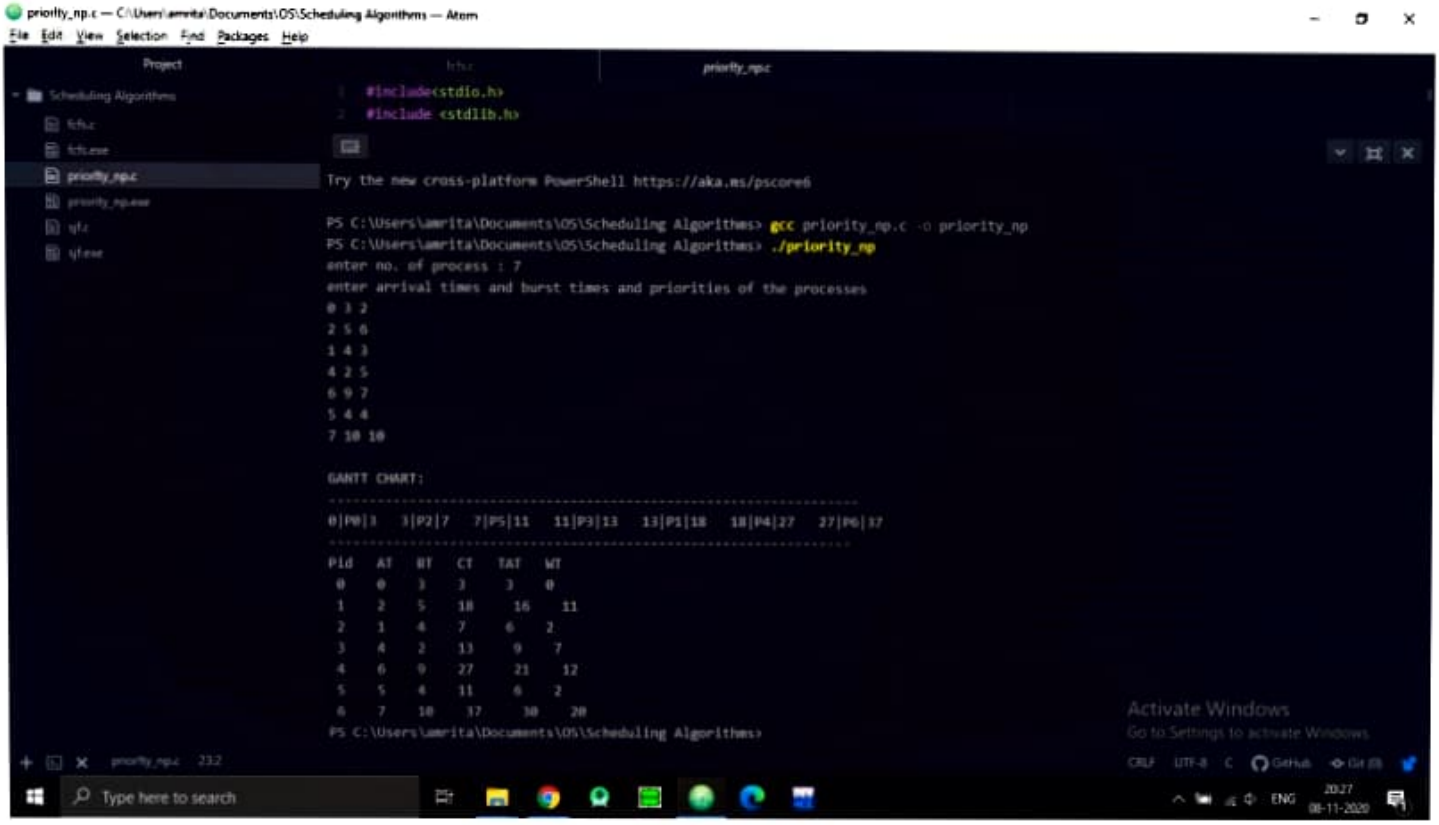
SOURCE CODE:

```
#include<stdio.h>
#include <stdlib.h>
int min(int a[], int priority[], int time, int n)
{
int i,j=-1;
int minimum=32625;
for(i=0;i<n;i++)
{
if(a[i]!=-1 && a[i]<=time)
{
if(priority[i]<minimum)
{
minimum=priority[i];
j=i;
}
}
}
return j;
}
```

```
int completeAll(int a[],int n)
{
int i;
for(i=0;i<n;i++)
{
if(a[i]!=-1)
return 0;
}
return 1;
}
void main()
{
int i,j,k,n;
printf("enter no. of process : ");
scanf("%d",&n);
int arrival[n],ct[n],bt[n],wt[n],turnAr[n],priority[n],arp[n];
printf("enter arrival times and burst times and priorities of the
processes \n");
for(i=0;i<n;i++)
{
scanf("%d%d%d",&arrival[i],&bt[i],&priority[i]);
arp[i]=arrival[i];
}
int time=0;
printf("\nGANTT
CHART:\n----------------------------------------------------------------\n
while(!completeAll(arrival,n))
{
k=min(arrival,priority,time,n);
if(k==-1)
{
printf("%d|idle|%d ",time,time+1);
time++;
continue;
}
else
{
ct[k]=time+bt[k];
printf("%d", time );
turnAr[k]=ct[k]-arrival[k];
wt[k]=turnAr[k]-bt[k];
```

```
time+=bt[k];
printf("|P%d|%d ",k,time );
arrival[k]=-1;
}
}
printf("\n----------------------------------------------------------------\n");
printf("Pid AT BT CT TAT WT\n");
for(i=0;i<n;i++)
{
printf(" %d %d %d %d %d
%d\n",i,arp[i],bt[i],ct[i],turnAr[i],wt[i] );
}
}
```

OUTPUT:



4. Implement Pre-emptive Priority Based job scheduling algorithm in C programming language.

SOURCE CODE:

```
#include<stdio.h>
#include <stdlib.h>
```

```
int min(int a[], int priority[], int time, int n)
{
int i,j=-1;
int minimum=32625;
for(i=0;i<n;i++)
{
if(a[i]!=-1 && a[i]<=time)
{
if(priority[i]<minimum)
{
minimum=priority[i];
j=i;
}
}
}
return j;
}
int completeAll(int a[],int n)
{
int i;
for(i=0;i<n;i++)
{
if(a[i]!=-1)
return 0;
}
return 1;
}
void main()
{
int i,j,k,n;
printf("enter no. of process : ");
scanf("%d",&n);
int arrival[n],ct[n],bt[n],wt[n],turnAr[n],priority[n],btun[n],arp[n];
printf("enter arrival times and burst times and priorities of the
processes \n");
for(i=0;i<n;i++)
{
scanf("%d%d%d",&arrival[i],&bt[i],&priority[i]);
btun[i]=bt[i];
arp[i]=arrival[i];
}
```

```
int time=0;
printf("\nGANTT
CHART:\n--------------------------------------------------------------------\n");
while(!completeAll(arrival,n))
{
k=min(arrival,priority,time,n);
if(k==-1)
{
printf("%d|idle|%d ",time,time+1);
time++;
continue;
}
else
{
printf("%d|P%d|%d ",time,k,time+1);
time++;
bt[k]--;
if(bt[k]==0)
{
ct[k]=time;
turnAr[k]=ct[k]-arrival[k];
wt[k]=turnAr[k]-btun[k];
arrival[k]=-1;
}
}
}
printf("\n--------------------------------------------------------------------\n");
printf("Pid AT BT CT TAT WT\n");
for(i=0;i<n;i++)
{
printf(" %d %d %d %d %d
%d\n",i,arp[i],btun[i],ct[i],turnAr[i],wt[i] );
}
}
```

OUTPUT:

```
fcfs.exe
prior_preemptive.c
prior_preemptive.exe
priority_np.c
priority_np.exe
sjf.c
sjf.exe
PS C:\Users\amrita\Documents\OS\Scheduling Algorithms> ./prior_preemptive
enter no. of process : 7
enter arrival times and burst times and priorities of the processes
0 1 2
1 7 6
2 3 3
3 6 5
4 5 4
5 15 10
6 8 9

GANTT CHART:
-----------------------------------------------------------------
0|P0|1   1|P1|2   2|P2|3   3|P2|4   4|P2|5   5|P4|6   6|P4|7   7|P4|8   8|P4|9   9|P4|10   10|P3|11   11|P3|12   12|P3|13   13|P3|14   1
4|P3|15   15|P3|16   16|P1|17   17|P1|18   18|P1|19   19|P1|20   20|P1|21   21|P1|22   22|P6|23   23|P6|24   24|P6|25   25|P6|26   26|P6
|27   27|P6|28   28|P6|29   29|P6|30   30|P5|31   31|P5|32   32|P5|33   33|P5|34   34|P5|35   35|P5|36   36|P5|37   37|P5|38   38|P5|39
  39|P5|40   40|P5|41   41|P5|42   42|P5|43   43|P5|44   44|P5|45
-----------------------------------------------------------------
Pid   AT   BT   CT   TAT   WT
 0    0    1    1     1    0
 1    1    7    22     21    14
 2    2    3    5     3    0
 3    3    6    16     13    7
 4    4    5    10     6    1
 5    5    15    45     40    25
 6    6    8    30     24    16
PS C:\Users\amrita\Documents\OS\Scheduling Algorithms>
```

SOURCE CODE:

```
#include <stdlib.h>
#include <stdio.h>
struct queue
{
int f;
int r;
int a[100];
};
void enque(struct queue *q,int x)
{
if(q->r==100)
{
printf("\nQUEUE OverFlow\n");
exit(0);
}
q->r+=1;
q->a[q->r]=x;
}
int deque(struct queue *q)
{
if(q->r==q->f)
return -1;
else
{
q->f+=1;
return q->a[(q->f)];
}
}
int addtoqueue(int arrival[],int bt[],int n,int time,struct queue *q)
{
int i,j;
int f=0;
for(i=0;i<n;i++)
{
if(arrival[i]!=-1 && time==arrival[i] && bt[i]>0)
{
f=1;
enque(q,i);
}
```

```
}
return f;
}
int completeAll(int a[],int n)
{
int i;
for(i=0;i<n;i++)
{
if(a[i]!=-1)
return 0;
}
return 1;
}
void main()
{
int i,j,k,n,x;
printf("enter no. of process : ");
scanf("%d",&n);
int arrival[n],ct[n],bt[n],wt[n],turnAr[n],arp[n],btun[n];
int maxAT=0;
printf("enter arrival times and burst times of the processes \n");
for(i=0;i<n;i++)
{
scanf("%d%d",&arrival[i],&bt[i]);
arp[i]=arrival[i];
if(arrival[i]>maxAT)
maxAT=arrival[i];
btun[i]=bt[i];
}
int TQ;
printf("enter quantum time : ");
scanf("%d",&TQ);
int time=0;
struct queue *q;
q=(struct queue*)malloc(sizeof(struct queue));
q->r=q->f=-1;
k=addtoqueue(arrival,bt,n,time,q);
while(k!=1)
{
time++;
k=addtoqueue(arrival,bt,n,time,q);
```

```
printf("%d\n", k);
}
while(!completeAll(arrival,n))
{
if(q->r==q->f)
{
time++;
if(time<=maxAT)
{
x=addtoqueue(arrival,bt,n,time,q);
while(x!=1)
{
time++;
if(time>50)
break;
x=addtoqueue(arrival,bt,n,time,q);
}
}
}
else
{
k=deque(q);
if(bt[k]>TQ)
{
bt[k]-=TQ;
for(i=1;i<=TQ;i++)
{
if(time+i<=maxAT)
x=addtoqueue(arrival,bt,n,time+i,q);
else
break;
}
enque(q,k);
time+=TQ;
}
else
{
for(i=1;i<=bt[k];i++)
{
if(time+i<=maxAT)
x=addtoqueue(arrival,bt,n,time+i,q);
```

```
else
break;
}
time+=bt[k];
ct[k]=time;
turnAr[k]=ct[k]-arrival[k];
wt[k]=turnAr[k]-btun[k];
bt[k]=0;
arrival[k]=-1;
}
}
if(time>50)
break;
}
printf("Pid AT BT CT TAT WT\n");
for(i=0;i<n;i++)
{
printf(" %d %d %d %d %d
%d\n",i,arp[i],btun[i],ct[i],turnAr[i],wt[i] );
}
}
```

OUTPUT: