```
//Gaurav Rai 1706019
//Higher Number Greater Priority
#include<bits/stdc++.h>

using namespace std;
class PCB
{
    public :
    static float avgWt,avgTat,avgCt;
    static int tq;
    int pid;
    int pri;
    bool vis;
    int bt,at,ct,tat,wt,tempbt;
    bool operator ==(const PCB p) const
    {
        if(at!=p.at || bt!=p.bt ||pid!=p.pid || pri!=p.pri)
        return false;
        return true;
    }
};
int PCB::tq =1;
float PCB::avgWt =0;
float PCB::avgTat =0;
float PCB::avgCt =0;
class less_than_at
{
    public:
    inline bool operator() (const PCB& struct1, const PCB& struct2)
    {
        return (struct1.at < struct2.at);
    }
};

class less_than_bt
{
    public:
    inline bool operator() (const PCB& struct1, const PCB& struct2)
    {
        return (struct1.bt < struct2.bt);
    }
};
class less_than_pid
{
    public:
    inline bool operator() (const PCB& struct1, const PCB& struct2)
    {
        return (struct1.pid < struct2.pid);
    }
};
```

```
class less_than_pri
{
    public:
    inline bool operator() (const PCB& struct1, const PCB& struct2)
    {
        if(struct1.pri==struct2.pri)
        return (struct1.at < struct2.at);
        return (struct1.pri > struct2.pri);
    }
};
void Priority(deque<PCB> &P,deque<pair<int ,PCB > > &ganttChart,int n,int tq)
{
    deque<PCB> readyQ;
    readyQ.push_back(P[0]);
    P[0].vis=true;
    int time=0;
    while(!readyQ.empty())
    {
        PCB temp=readyQ.front();
        readyQ.pop_front();
        time+=tq;
        ganttChart.push_back({time,temp});
        for(int i=0;i<P.size();i++)
        {
            if(P[i].at<=time && P[i].vis==false)
            {readyQ.push_back(P[i]); P[i].vis=true;}
        }
        if(temp.tempbt+tq<temp.bt)
        {
            temp.tempbt+=tq;
            readyQ.push_back(temp);
        }
        sort(readyQ.begin(),readyQ.end(),less_than_pri());

    }
    P.clear();
    int countt=0;
    for(auto it=ganttChart.rbegin();countt<n;it++)
    {
        PCB temp=it->second;
        if(!(find(P.begin(),P.end(),temp)!=P.end()))
        {temp.ct=it->first; temp.tat=temp.ct-temp.at;temp.wt=temp.tat-temp.bt;
P.push_back(temp); countt++;}
    }
    cout<<"Gantt Chart : - > "<<endl;
    cout<<"PID :  |";
    for(auto e :ganttChart)
    {
        cout<<"P"<<e.second.pid<<" | ";
    }
```

```
    cout<<endl<<"Time : |";
    for(auto e :ganttChart)
    {
        cout<<setw(2)<<e.first<<" | ";
    }
    cout<<endl;

}

int main()
{
    int n;
    cout<<"Enter the number of processes ...";cin>>n;
    deque<PCB> P(n);
    for(int i=0;i<n;i++)
    {
        cout<<"Process Id : ";cin>>P[i].pid;
        cout<<"Arrival Time : ";cin>>P[i].at;
        cout<<"Burst Time : ";cin>>P[i].bt;
        cout<<"Priority : ";cin>>P[i].pri;
        P[i].vis=false;
        P[i].tempbt=0;
    }
    sort(P.begin(),P.end(),less_than_at());
    deque<pair<int,PCB> > ganttChart;
    Priority(P,ganttChart,n,PCB::tq);
    for(int i=0;i<P.size();i++)
    {
        PCB::avgWt +=P[i].wt;
        PCB::avgTat+=P[i].tat;
        PCB::avgCt+=P[i].ct;
    }
    PCB::avgWt/=P.size();
    PCB::avgTat/=P.size();
    PCB::avgCt/=P.size();
    sort(P.begin(),P.end(),less_than_pid());
    cout<<"\tPriority(Preemptive) CPU SCHEDULING\n";
    cout<<"PID\tPri\tAT\tBT\tCT\tTAT\tWT\n";
    for(int i=0;i<P.size();i++)

cout<<P[i].pid<<"\t"<<P[i].pri<<"\t"<<P[i].at<<"\t"<<P[i].bt<<"\t"<<P[i].ct<<"\t"<<
P[i].tat<<"\t"<<P[i].wt<<endl;
    cout<<"\tAverage Waiting Time : "<<PCB::avgWt<<endl;
    cout<<"\tAverage TurnAround Time : "<<PCB::avgTat<<endl;
    cout<<"\tAverage Completion Time : "<<PCB::avgCt<<endl;

    return 0;
}
```