Activity Selection Priority Queue Based on Start Time

Given a set ‘S’ of ‘n’ activities, implement the recursive greedy algorithm to select a subset of activities from S by selecting the task that starts last. Build a priority queue (max heap) based on start time and then select the activities

Input Format

First line contains the number of activities, n

Next 'n' line contains the details of the activities such as name of activity, start time and finish time

Output Format

Print the name of the activities that are selected separated by a space

```
#include<iostream>
#include<queue>
#include<set>
using namespace std;
class activity
{
   public:
   string name;
   int s,f;
   friend istream& operator>>(istream&,activity&);
   friend ostream& operator<<(ostream&,const activity&);
   bool operator<(const activity&) const;
};
istream& operator>>(istream& in,activity& a)
{
   in>>a.name>>a.s>>a.f;
}
ostream& operator<<(ostream& out,const activity& a)
{
   out<<a.name<<" ";
}
bool activity::operator<(const activity& a) const
{
   return a.s>s;
}
void recursive_activity_selector(priority_queue<activity> &pq,vector<activity> &v)
{
   activity a=pq.top();
   pq.pop();
   v.push_back(a);
   while(!pq.empty() && pq.top().f>a.s)
   {
       pq.pop();
   }
   if(!pq.empty())
   {
       recursive_activity_selector(pq,v);
   }
   else
   {
       return;
   }
}
int main()
{
   priority_queue<activity> pq;
   vector<activity> v;
   activity a;
   int n;
   cin>>n;
   for(int i=0;i<n;i++)
   {
       cin>>a;
       pq.push(a);
   }
   recursive_activity_selector(pq,v);
   for(auto i=v.end()-1;i!=v.begin()-1;i--)
   {
       cout<<*i;
   }
   return 0;
}
```

Activity Selection using Priority Queue based on Start time

Given a set ‘S’ of ‘n’ activities, implement the iterative greedy algorithm to select a subset of activities from S by selecting the task that starts last. Build a priority queue (max heap) based on start time and then select the activities

Input Format

First line contains the number of activities, n

Next 'n' line contains the details of the activities such as name of activity, start time and finish time

Output Format

Print the name of the activities that are selected separated by a space

```
#include<iostream>
#include<queue>
#include<set>
using namespace std;
class activity
{
   public:
   string name;
   int s,f;
   friend istream& operator>>(istream&,activity&);
   friend ostream& operator<<(ostream&,const activity&);
   bool operator<(const activity&) const;
};
istream& operator>>(istream& in,activity& a)
{
   in>>a.name>>a.s>>a.f;
}
ostream& operator<<(ostream& out,const activity& a)
{
   out<<a.name<<" ";
}
bool activity::operator<(const activity& a) const
{
   return a.s>s;
}
void iterative_activity_selector(priority_queue<activity> &pq,vector<activity> &v)
{
   activity a=pq.top();
   pq.pop();
   v.push_back(a);
   while(!pq.empty())
   {
      if(pq.top().f<=a.s)
      {
         a=pq.top();
```

```
            v.push_back(a);
        }
        pq.pop();
    }
}
int main()
{
    priority_queue<activity> pq;
    vector<activity> v;
    activity a;
    int n;
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cin>>a;
        pq.push(a);
    }
    iterative_activity_selector(pq,v);
    for(auto i=v.end()-1;i!=v.begin()-1;i--)
    {
        cout<<*i;
    }
    return 0;
}
```

Activity Selection Sort Recursive

Given a set ‘S’ of ‘n’ activities, implement the recursive greedy algorithm to select a subset of activities from S by selecting the task that finishes first. Sort the activities in ascending order based on finsih time and then select the activities

Input Format

First line contains the number of activities, n

Next 'n' line contains the details of the activities such as name of activity, start time and finish time

Output Format

Print the name of the activities that are selected separated by a space

```
#include<iostream>
```

```
#include<queue>
#include<set>
using namespace std;
class activity
{
    public:
    string name;
    int s,f;
    friend istream& operator>>(istream&,activity&);
    friend ostream& operator<<(ostream&,const activity&);
    bool operator<(const activity&) const;
};
istream& operator>>(istream& in,activity& a)
{
    in>>a.name>>a.s>>a.f;
}
ostream& operator<<(ostream& out,const activity& a)
{
    out<<a.name<<" ";
}
bool activity::operator<(const activity& a) const
{
    return a.f<f;
}
void recursive_activity_selector(priority_queue<activity> &pq,vector<activity> &v)
{
    activity a=pq.top();
    pq.pop();
    v.push_back(a);
    while(!pq.empty() && pq.top().s<a.f)
    {
        pq.pop();
    }
    if(!pq.empty())
    {
        recursive_activity_selector(pq,v);
    }
    else
    {
        return;
    }
}
int main()
{
    priority_queue<activity> pq;
    vector<activity> v;
    activity a;
    int n;
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cin>>a;
        pq.push(a);
    }
    recursive_activity_selector(pq,v);
    for(auto a:v)
    {
        cout<<a;
    }
    return 0;
}
```