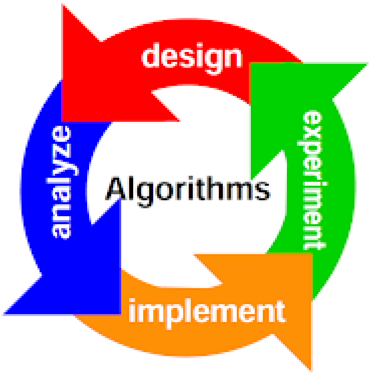
সেগমেন্টেড সিভ (Segmented Sieve) হল নির্দিষ্ট রেঞ্জ এর মধ্যে প্রাইম নম্বর জেনারেট করার জন্য খুব কার্যকর এলগরিদম।



যদি A এবং B এর মধ্যে প্রাইম নম্বর জেনারেট করতে বলা হয় যেখানে (A-B) এর পার্থ্যক্য 10^5   এর মধ্যে থাকবে এবং B এর সর্বোচ্চ লিমিট 10^12 হবে সেক্ষেত্রে সেগমেন্টেড সিভ দিয়ে প্রাইম জেনারেট করা যায়।

যখন 10^8 এর উপর প্রাইম নম্বর জেনারেট করতে বলা হয় তখন নরমাল সিভ এলগরিদম কাজ করবে না। কারন এরের সর্বোচ্চ সাইজ অতিক্রম করে যাবে। সেক্ষেত্রে সেগমেন্টেড সিভ কাজে দিবে। কারন 10^12   প্রাইম কিনা তা চেক করার জন্য 10^6   পর্যন্ত প্রাইম নম্বর জেনারেট করলেই চলবে। এরপর প্রাইম নম্বর গুলোর কম্পোজিট নম্বর 10^12 পর্যন্ত বের করে তাদের বাদ দিলেই বাকি নম্বরগুলোই হবে প্রাইম নম্বর। নিচের উদাহরনে ব্যাপারটা সহজে বোঝা যাবে।

ধরা যাক ২৯ থেকে ৭০০ পর্যন্ত প্রাইম নম্বর জেনারেট করব। এখানে A=29 B=700 ।

১ প্রথমে সিভ এলগরিদম দিয়ে   ৭০০ এর স্কয়ার রুট= ২৬ পর্যন্ত প্রাইম নম্বরগুলো জেনারেট করে নিব।

p[0]=2
p[1]=3
p[2]=5
p[3]=7
p[4]=11
p[5]=13
p[6]=17
p[7]=19

p[8]=23

২ এখন এই প্রাইম নম্বরগুলো দিয়েই ৭০০ পর্যন্ত প্রাইম নম্বর বের করা যাবে। ২৯ থেকে এই প্রাইম গুলোর যত কম্পোজিট নম্বর আছে সব বের করে বাদ দিলেই কাজ শেষ ।

৩ সবার আগে একটি এরেতে ২৯ থেকে ৭০০ পর্যন্ত সব নম্বর রেখে দেই।

```
for(i=l;i<=u;i++)
{
    sg.push_back(i);
 }
```

৪ যেহেতু লো লিমিট ২৯ থেকে শুরু করব এবং ২ থেকে ২৩ পর্যন্ত সকল প্রাইম নম্বর ব্যবহার করব তাই এবার i=0 থেকে sqrt(700)=২৬ পর্যন্ত একটা লুপ চালাই যেখানে এইপ্রাইম নম্বর গুলোর কম্পোজিট নম্বরগুলো কে মার্ক করা হবে। লুপ এর ভেতর প্রত্যেক প্রাইম নম্বর (যা সিভ থেকে জেনারেট করা) একটি ভ্যারিয়াবলে রেখে কম্পোজিট নম্বর বের করব । এখানে একটি সূত্র আছে । আমাদের start point টি বা রেঞ্জ এর প্রথম কম্পোজিট নম্বর টি কোথা থেকে শুরু হবে তা এই সূত্র এর মাধ্যমে বের করব।

```
   for(i=0;p[i]<=root;i++)
   {
       si=p[i];

       start=si*si;   //প্রত্যেক প্রাইম এর স্কয়ার কম্পোজিট হয়।

       if(start<l)
       {
           start=((l+si-1)/si)*si;   //  রেঞ্জ এর মধ্যে প্রথম কম্পোজিট নম্বর বের করার জন্য
       }
}
```

৫ এবার start point থেকে লুপ চালাবো B অর্থাৎ ৭০০ পর্যন্ত। যদি j=start point থেকে শুরু হয় তাহলে প্রতিবার লুপ j=j+prime number (si) করে ইন্ক্রিমেন্ট করব তাহলে সেই প্রাইম এর সকল কম্পোজিট বের হয়ে যাবে।

```
 for(i=0;p[i]<=root;i++)
    {
        si=p[i];

        start=si*si;

        if(start<l)
        {
            start=((l+si-1)/si)*si;
        }


        for(j=start;j<=u;j=j+si)
        {
            sg[j-l]=0;   // শুরু থেকে ইন্ডেক্স করার জন্য।

        }
    }
```

যেমনঃ লুপ এ প্রথমে si=p[0]=2

start=2*2=4

4<29

start=((29+2-1)/2)*2
=15*2
=30

প্রথম কম্পোজিট ৩০

j=30   থেকে

sg[30-29]=sg[1]=0;

sg[0] index হল ২৯ সবার শুরুতে কোড লিখেছি। sg[1] 30 ছিল যা এখন 0 করে দিয়েছি।

এরপর j=30+2=32,34,36,38......................700 এই সিরিজের সব নম্বর কম্পোজিট হিসেবে মার্ক হবে।

তারপর এই লুপ শেষ হবে এবং শুরুর লুপ এ i=1 হবে। তখন si=p[1]=3

এভাবে 30+3=33,36,39...... এই সিরিজের সব নম্বর বাদ যাবে

এভাবে ২৩ পর্যন্ত সব প্রাইম নম্বর দিয়ে কম্পোজিট নম্বর বের করে বাদ দিয়ে দিবো। যা থাকবে সেগুলো নম্বর এ প্রাইম হবে। পুরো কোড দেয়া হল নিচে।

```
#include <bits/stdc++.h>
using namespace std;
#define n 10000000
vector<long long int>s,sg,segment;
long long int p[n],k=1,size;
bool a[n];
long long int prime()
{
    long long i,j;
    a[0]=a[1]=1;
    for(i=4;i<=n;i=i+2)
    {
        a[i]=1;
    }
    for(i=3;i<=sqrt(n);i=i+2)
    {
        for(j=i*i;j<=n;j=j+2*i)
        {
            a[j]=1;
        }
    }
    p[0]=2;
    for(i=3;i<=n;i=i+2)
    {
        if(a[i]==0)
        {
            p[k]=i;
            //cout<<p[k]<<endl;
            k++;
        }
    }


}
```

```
void segmented_sieve(long long int l,long long int u)
{
    long long int root,start,i,j,si;
    sg.clear();
    root=sqrt(u)+1;



    for(i=l;i<=u;i++)
    {
        sg.push_back(i);
    }


    if(l==0)
    {
        sg[1]=0;
    }
    else if(l==1)
    {
        sg[0]=0;
    }



    for(i=0;p[i]<=root;i++)
    {
        si=p[i];

        start=si*si;

        if(start<l)
        {
            start=((l+si-1)/si)*si;
        }


        for(j=start;j<=u;j=j+si)
        {
            sg[j-l]=0;

        }
```

```
    }

}

int main()
{
     long long int m,g,c,r,t,l,h,u,w,tc,tx,i,j;

     prime();
     cin>>tc;

     for(tx=1;tx<=tc;tx++)
     {

        cin>>l>>u;
        segmented_sieve(l,u);

        for(i=l;i<=u;i++)
        {


            if(sg[i-l]!=0)
            {
                segment.push_back(sg[i-l]);
            }
        }

        for(i=0;i<segment.size();i++)
        {
            cout<<segment[i]<<endl;
        }
        segment.clear();
        sg.clear();
        cout<<endl;



     }



    return 0;
}
```