Bài toán: Hãy liệt kê tất cả các số nguyên tố trong đoạn [1,*N*].
Đây là bài toán cơ bản. Mục đích chúng ta xây dựng:
int Prime[maxN] - Mảng các số nguyên tố Prime[1]=2, Prime[2]=3, ...
int nP - số lượng các số nguyên tố tìm được
int factor[maxN] - Cho thừa số nguyên tố nhỏ nhất, ví dụ factor[10]=2
Mệnh đề: nếu thừa số nguyên tố nhỏ nhất của a là p thì với mọi số nguyên tố x nhỏ hơn hoặc bằng p ta có thừa số nguyên tố nhỏ nhất của a.x là x. Do đó ta có thuật toán sau:
Khởi đầu mảng factor bằng 0, bắt đầu duyệt từ số 2 đến số n. Nếu như đến số i ta có factor[i]=0 thì i là số nguyên tố và ta thêm i vào mảng Prime đồng thời đặt thừa số nguyên tố nhỏ nhất của i là i. Tiếp theo, duyệt tất cả các số nguyên tố nhỏ hơn hoặc bằng thừa số nguyên tố nhỏ nhất của i giả sử số x. Khi đó đặt thừa số nguyên tố nhỏ nhất của x*i là x:

```
biến chính: int nP, Prime[maxn], factor[maxn];

void sangNT(int n) {
   for(int i=2;i<=n;i++) factor[i]=0;
   nP=0;
   for(int i=2;i<=n;i++) {
      if (factor[i]==0) {Prime[++nP]=i; factor[i]=nP;}
      int j=1;
      while (j<=nP && Prime[j]<=factor[i] && Prime[j]<=n/i) {
         factor[Prime[j]*i]=Prime[j];
         j++;
      }
   }
}
```

Chú ý sản phẩm của thủ tục trên là:

- Số nguyên **nP** - số lượng các số nguyên tố thuộc [1,n]
- Mảng **int Prime[maxn]** - mảng các số nguyên tố (2, 3, 5, 7, ...)
- Mảng **int factor[maxn]** - mảng chứa thừa số nguyên tố nhỏ nhất của một số.

Như vậy một số p là số nguyên tố khi và chỉ khi **factor[p]==p.** Ví dụ với số 7 thì factor[7]=7.
Chú ý rằng trên mảng factor, mỗi số chỉ thay đổi giá trị đúng một lần. Do đó độ phức tạp của thuật toán trên là O(n)